20 TOR.
10 INTL
10 NANC
5 OTT 5 CAL.

5 BOXES
50

EMBEDDED CONTROL

H A N D B O O K



1994/95

MICROCHIP



Embedded Control Handbook

Serving A Complex And Competitive World With Field-Programmable Embedded Control System Solutions

© 1994 Microchip Technology Inc.

September 1994 / DS00092C

"Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Inc. with respect to the accuracy or use of such information, or infringement of patents arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights."

PIC is a registered trademark of Microchip Technology Inc. in the U.S.A.
The Microchip logo and name are trademarks of Microchip Technology Incorporated.
The Embedded Control Solutions Company is a trademark of Microchip Technology Inc.
*fuzzy*LAB, PICMASTER, PRO MATE, PICSTART, PICSEE, PICPRO, TrueGauge, Total Endurance,UniMouse, SEEVAL, Smart Serial and SQTP are trademarks of Microchip Technology Inc.
All rights reserved. Copyright © 1994, Microchip Technology Inc.

ALLPRO is a trademark of Logical Devices, Inc.
Apple and Macintosh are registered trademarks of Apple Computer, Inc.
Apple Desktop Bus is a trademark of Apple Computer, Inc.
CompuServe is a registered trademark of CompuServe, Inc.
Data I/O is a registered trademark of Data I/O Corp.
DSPLAY is a trademark of Burr Brown Corp.
DFDP is a trademark of Atlanta Signal Processing Inc.
*fuzzy*TECH is a registered trademark of INFORM Software Corp.
Hatachi is a registered trademark of Hatachi.
I²C is a trademark of Philips Corp.
IBM, IBM PC, PC/XT and AT are registered trademarks of IBM Corp.
INFORM is a trademark of INFORM Software Corp.
Intel is a trademark of Intel Corp.
MathCad is a trademark of MathSoft Inc.
Excel, Microsoft, Mouse Systems, MS-DOS are registered trademarks of Microsoft Corp.
Microwire and Tri-State are trademarks of National Semiconductor.
Mouse Systems is a registered trademark of MSC Technologies, Inc.
PROCOMM is a registered trademark of Datastorm Technologies, Inc.
SMC is a trademark of Standard Microsystems Corp.
Unisite, PROMLINK, PROCOMM, SITE, Site 48, ChipSite, PinSite, SetSite, HiTerm and HandlerSite are trademarks of Data I/O Corporation.
Windows is a trademark of Microsoft Corp.

All other trademarks mentioned herein are the property of their respective companies.

Table of Contents

SECTION 1	INTRODUCTION TO EMBEDDED SOLUTIONS FROM MICROCHIP	PAGE
	Embedded Control Overview	1- 1
	PIC16/17 Families Overview and Road Map	1- 1
	PIC16C5X: Base-Line Family	1- 1
	PIC16CXX: Mid-Range Family	1- 1
	PIC17CXX: High-End Family	1- 3
	PIC16/17 Naming Convention	1- 4
	Serial EEPROM Overview	1- 6
	OTP EPROM Overview	1- 6
	The Advantages of One-Time-Programmable	1- 6
	Application Specific Standard Products	1- 8
	Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP™)	1- 8
	What's New in the 1994/95 Embedded Control Handbook	1- 10
	What's Changed in the 1994/95 Embedded Control Handbook	1- 10

SECTION 2	PIC16C5X APPLICATION NOTES	
	A Comparison of Low End 8-Bit Microcontrollers - AN520	2- 1
	Power-Up Considerations - AN522	2- 11
	Software Interrupt Techniques - AN514	2- 15
	Software Stack Management - AN527	2- 19
	Implementing Long Calls - AN581	2- 23
	Macros for Page and Bank Switching - AN586	2- 27
	Implementing Wake-Up on Keystroke - AN528	2- 47
	Interfacing to AC Power Lines - AN521	2- 51
	Frequency Counter Using PIC16C5X - AN592	2- 53
	Analog to Digital Conversion - AN513	2- 59
	Implementing Ohmeter/Temperature Sensor - AN512	2- 65
	Implementing a Simple Serial Mouse Controller - AN519	2- 71
	Intelligent Remote Positioner - AN531	2- 83
	A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs - AN590	2- 99
	Multiplexing LED Drive and a 4 x 4 Keypad Sampling - AN529	2-109
	Using PIC16C5X Microcontrollers as LCD Drivers - AN563	2-133
	PLD Replacement - AN511	2-145
	Serial Port Routines Without Using the RTCC - AN593	2-165
	Implementation of an Asynchronous Serial I/O - AN510	2-171
	Using PIC16C5X as a Smart I²C™ Peripheral - AN541	2-193
	PIC16C54A EMI Results - AN577	2-207

SECTION 3	PIC16CXX APPLICATION NOTES	
	Using the PortB Interrupt on Change as an External Interrupt - AN566	3- 1
	Implementing Wake Up on Keystroke - AN552	3- 5
	Using the 8-Bit Parallel Slave Port - AN579	3- 9
	A PC-Based Development Programmer for the PIC16C84 - AN589	3- 15
	Using the CCP Modules - AN594	3- 21
	Interfacing to an LCD Module - AN587	3- 49
	Using Timer1 in Asynchronous Clock Mode - AN580	3- 95
	Low-Power Real Time Clock - AN582	3- 99
	Using the Analog to Digital Converter - AN546	3-123
	Four Channel Digital Voltmeter with Display and Keyboard - AN557	3-141
	Apple® Desktop Bus - AN591	3-169
	Software Implementation of Asynchronous Serial I/O - AN555	3-181
	Software Implementation of I²C™ Bus Master - AN554	3-223
	Use of the SSP Module in the I²C Multi-Master Environment - AN578	3-297

© 1994 Microchip Technology Inc.

MICROCHIP

SECTION 4	PIC17CXX APPLICATION NOTES	PAGE
	Saving and Restoring Status on Interrupt - AN534	4- 1
	Implementing Table Read and Table Write - AN548	4- 3
	Frequency and Resolution Options for PWM Outputs - AN539	4- 7
	Using PWM to Generate Analog Output - AN538	4- 15
	Using the PWM - AN564	4- 17
	Using the Capture Module - AN545	4- 29
	Serial Port Utilities - AN547	4- 53
	Math Utility Routines - AN544	4- 63
	Implementing IIR Digital Filters - AN540	4-129
	Implementation of Fast Fourier Transforms - AN542	4-145
	Tone Generation - AN543	4-163
	Servo Control of a DC Brush Motor - AN532	4-185
	Implementation of the Data Encryption Standard Using PIC17C42 - AN583	4-273
SECTION 5	PIC16/17 APPLICATION NOTES	
	Implementing a Table Read - AN556	5- 1
	Techniques to Disable Global Interrupts - AN576	5- 5
	IEEE 754 Compliant Floating-Point Routines - AN575	5- 11
	PIC16C5X / 16CXX Utility Math Routines - AN526	5- 89
	A Real-Time Operating System for PIC16/17 - AN585	5-167
	PIC16/17 Oscillator Design Guide - AN588	5-213
	PICMASTER™ Support of Microsoft® Windows™ DDE - AN584	5-231
SECTION 6	ASSP PRODUCTS APPLICATION NOTES	
	Calibrating the MTA11200 System - AN570	6- 1
	Communicating with EEPROM in MTA8XXXX - AN571	6- 21
	Hardware and Software Resolution for a Pointing Device - AN569	6- 33
SECTION 7	INTERFACING PIC16/17 WITH SERIAL EEPROMS	
	Communicating with I²C Bus Using the PIC16C5X - AN515	7- 1
	Interfacing 93CX6 Serial EEPROMs to the PIC16C5X Microcontrollers- AN530	7- 11
	Logic Powered Serial EEPROMs - AN535	7- 29
SECTION 8	SERIAL EEPROMS TUTORIALS AND APPLICATION NOTES	
	Basic Serial EEPROM Operation - AN536	8- 1
	Everything a System Engineer Needs to Know About Serial EEPROM Endurance - AN537	8- 15
	Using the Microchip Endurance Predictive Software - AN562	8- 23
	Interfacing 24LCXXB Serial EEPROMs to the PIC16C54 - AN567	8- 27
	Using the 24XX65 and 24XX32 with Stand-alone PIC16C54 Code - AN558	8- 53
	24C01A Compatibility Issue and Its Mobility for Memory Upgrade - AN517	8- 93
	Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559	8- 95
	Using the 93LC56 and 93LC66 - AN560	8- 99
	1.8 Volt Technology - Benefits - AN550	8-119
	Serial EEPROM Solutions vs. Parallel Solutions - AN551	8-121
	Questions and Answers Concerning Serial EEPROMs - AN572	8-125

4

5

6

7

8

© 1994 Microchip Technology Inc.

MICROCHIP

Table of Contents

SECTION 9	DEVELOPMENT TOOLS	PAGE
	Development System Selection Chart	9- 1
	Microchip Bulletin Board Service (BBS)	9- 3
	Systems Information Line	9- 5
	Application Specific Standard Products Tools:	
	PICSEE™ Product Development Tools	9- 7
	TrueGauge™ Intelligent Battery Management Development Tool	9- 11
	Logic Product Tools:	
	PICMASTER™ Product Brief	9- 17
	PRO MATE™ Product Brief	9- 21
	PICSTART™-16B1 Product Brief	9- 25
	PICSTART™-16C Product Brief	9- 27
	PICDEM-1 Product Brief	9- 29
	PICDEM-2 Product Brief	9- 31
	MPASM Universal Assembler Product Brief	9- 33
	MPSIM Simulator Product Brief	9- 35
	MP-C Product Brief	9- 37
	*fuzzy*Tech®-MP Product Brief	9- 39
	Memory Products Tools:	
	Total Endurance™ Serial EEPROM Endurance Model	9- 41
	Designer's Kit	9- 43

SECTION 10	SELECTED READING OF MICROCHIP ARTICLES	
	Selected Reading	10- 1

SECTION 11	OFFICE LOCATIONS	
	Factory Representatives	11- 1
	Distributors	11- 7
	Factory Sales	11-17

9

10

11

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL

		Page
24C01A Compatibility Issues and Its Mobility for Memory Upgrade	AN517	8- 93
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs	AN590	2- 99
Analog to Digital Conversion	AN513	2- 59
A PC-Based Development Programmer for the PIC16C84	AN589	3- 15
Apple Desktop Bus	AN591	3-169
Basic Serial EEPROM Operation	AN536	8- 1
Calibrating the MTA11200	AN570	6- 1
Communicating with EEPROM in MTA85XXXX	AN571	6- 21
Communicating with I²C Bus Using PIC16C5X	AN515	7- 1
Comparison of 8-Bit Microcontrollers	AN520	2- 1
Disabling Global Interrupts	AN576	5- 5
Everything a System Engineer Needs to Know About Serial EEPROM Endurance	AN537	8- 15
IEEE 754 Compliant Floating Point Routines	AN575	5- 11
Four Channel Digital Volt Meter with Display and Keyboard	AN557	3-141
Frequency and Resolution Options for PWM Outputs	AN539	4- 7
Frequency Counter Using PIC16CXX	AN592	2- 53
Hardware and Software Resolution for a Pointing Device	AN569	6- 33
Implementation of the Data Encryption Standard Using PIC17C42	AN583	4-273
Implementing a Table Read	AN556	5- 1
Implementing a Simple Serial Mouse Controller	AN519	2- 71
Implementing an LCD Controller	AN563	2-133
Implementing IIR Digital Filters	AN540	4-129
Implementing Long Calls	AN581	2- 23
Implementing Ohmmeter/Temperature Sensor	AN512	2- 65
Implementing Table Read and Table Write	AN548	4- 3
Implementing Wake-Up on Keystroke	AN528	2- 47
Implementing Wake Up on Keystroke	AN552	3- 5
Implementation of an Asynchronous Serial I/O	AN510	2-171
Implementation of Fast Fourier Transforms	AN542	4-145
Intelligent Remote Positioner	AN531	2- 83
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X Microcontrollers	AN530	7- 11
Interfacing 24LCXX Serial EEPROMs to the PIC16C54	AN567	8- 27
Interfacing to AC Power Lines	AN521	2- 51
Interfacing to an LCD Module	AN587	3- 49
Logic Powered Serial EEPROMs	AN535	7- 29
Low Power Clock	AN591	3-169
Macros for Page and Bank Switching	AN586	2- 27
Math Utility Routines (PIC17CXX)	AN544	4- 63
Multiplexing LED Drive and a 4 x 4 Keypad Sampling	AN529	2-109
1.8 Volt Technology - Benefits	AN550	8-119
Optimizing Serial Bus Operations with Proper Write Cycle Times	AN559	8- 95
PIC16/17 Oscillator Design Guide	AN588	5-213
PIC16C54A EMI Results	AN577	2-207
PIC16C5X / 16CXX Utility Math Routines	AN526	5- 89
PICMASTER Support of Microsoft Windows' DDE	AN584	5-231
PLD Replacement	AN511	2-145
PortB as External Interrupt	AN566	3- 1
Power-Up Considerations	AN522	2- 11
Questions and Answers Concerning Serial EEPROMs	AN572	8-125
Real Time Operating System for PIC16/17	AN585	5-167
Saving and Restoring Status on Interrupt	AN534	4- 1
Serial EEPROM Solutions vs. Parallel Solutions	AN551	8-121
Serial Port Routines Without Using the RTCC	AN593	2-165

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL (continued)

		Page
Serial Port Utilities	AN547	4- 53
Servo Control of a DC Brush Motor	AN532	4-185
Software Implementation of Asynchronous Serial I/O	AN555	3-181
Software Implementation of I²C Bus Master	AN554	3-223
Software Interrupt Techniques	AN514	2- 15
Software Stack Management	AN527	2- 19
Tone Generation	AN543	4-163
Use of the SSP Module in the I²C Multi-Master Environment	AN578	3-297
Using PIC16C5X as a Smart I²C Peripheral	AN541	2- 19
Using PWM to Generate Analog Output	AN538	4- 15
Using the 8-Bit Parallel Slave Port	AN579	3- 9
Using the Capture Module	AN545	4- 29
Using the CCP Module	AN594	3- 21
Using the PWM	AN564	4- 17
Using the Analog to Digital Converter	AN546	3-123
Using the Microchip Endurance Predictive Software	AN562	8- 23
Using the 24XX65 and 24XX32 with Stand-alone PIC16/17 Code	AN558	8- 53
Using the 93LC56 and 93LC66	AN560	8- 99
Using Timer1 in Asynchronous Clock Mode	AN580	3- 95

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL

		Page
AN510	Implementation of an Asynchronous Serial I/O	2-171
AN511	PLD Replacement	2-145
AN512	Implementing Ohmmeter/Temperature Sensor	2- 65
AN513	Analog to Digital Conversion	2- 59
AN514	Software Interrupt Techniques	2- 15
AN515	Communicating with I²C Bus Using PIC16C5X	7- 1
AN517	24C01A Compatibility Issues and Its Mobility for Memory Upgrade	8- 93
AN519	Implementing a Simple Serial Mouse Controller	2- 71
AN520	Comparison of 8-Bit Microcontrollers	2- 1
AN521	Interfacing to AC Power Lines	2- 51
AN522	Power-Up Considerations	2- 11
AN526	PIC16C5X/16CXX Math Utility Routines	5- 89
AN527	Software Stack Management	2- 19
AN528	Implementing Wake-Up on Keystroke	2- 47
AN529	Multiplexing LED Drive and a 4 x 4 Keypad Sampling	2-109
AN530	Interfacing 93CX6 Serial EEPROMs to the PIC16C5X	7- 11
AN531	Intelligent Remote Positioner	2- 83
AN532	Servo Control of a DC Brush Motor	4-185
AN534	Saving and Restoring Status on Interrupt	4- 1
AN535	Logic Powered Serial EEPROMs	7- 29
AN536	Basic Serial EEPROM Operation	8- 1
AN537	Everything a System Engineer Needs to Know About Serial EEPROM Endurance	8- 15
AN538	Using PWM to Generate Analog Output	4- 15
AN539	Frequency and Resolution Options for PWM Outputs	4- 7
AN540	Implementing IIR Digital Filters	4-129
AN541	Using PIC16C5X as a Smart I²C Peripheral	2-193
AN542	Implementation of Fast Fourier Transforms	4-145
AN543	Tone Generation	4-163
AN544	Math Utility Routines	4- 63
AN545	Using the Capture Module	4- 29
AN546	Using the Analog to Digital Converter	3-123
AN547	Serial Port Utilities	4- 53
AN548	Implementing Table Read and Table Write	4- 3
AN550	1.8 Volt Technology - Benefits	8-119
AN551	Serial EEPROM Solutions vs. Parallel Solutions	8-121
AN552	Implementing Wake Up on Keystroke	3- 5
AN554	Software Implementation of I²C Bus Master	3-223
AN555	Software Implementation of Asynchronous Serial I/O	3-181
AN556	Implementing a Table Read	5- 1
AN557	Four Channel Digital Volt Meter with Display and Keyboard	3-141
AN558	Using the 24XX65 and 24XX32 with Stand-alone PIC16/17 Code	8- 53
AN559	Optimizing Serial Bus Operations with Proper Write Cycle Times	8- 95
AN560	Using the 93LC56 and 93LC66	8- 99
AN562	Using the Microchip Endurance Predictive Software	8- 23
AN563	Implementing an LCD Controller	2-133
AN564	Using the PWM	4- 17
AN566	PortB as External Interrupt	3- 1
AN567	Interfacing 24LCXX Serial EEPROMs to the PIC16C54	8- 27
AN569	Hardware and Software Resolution for a Pointing Device	6- 33
AN570	Calibrating the MTA11200	6- 1
AN571	Communicating with EEPROM in MTA85XXX	6- 21
AN572	Questions and Answers Concerning Serial EEPROMs	8-125
AN575	IEEE 754 Compliant Floating Point Routines	5- 11

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL (continued)

		Page
AN576	Disabling Global Interrupts	5- 5
AN577	PIC16C54A EMI Results	2-207
AN578	Use of the SSP Module in the I²C Multi-Master Environment	3-297
AN579	Using the 8-Bit Parallel Slave Port	3- 9
AN580	Using Timer1 in Sleep	3- 95
AN581	Implementing Long Calls	2- 23
AN582	Low Power Clock	3- 99
AN583	Implementation of the Data Encryption Standard Using PIC17C42	4-273
AN584	PICMASTER Support of Microsoft Windows' DDE	5-231
AN585	Real Time Operating System	5-167
AN586	Macros for Page and Bank Switching	2- 27
AN587	Interfacing to an LCD Module	3- 49
AN588	PIC16/17 Oscillator Design Guide	5-213
AN589	A PC-Based Development Programmer for the PIC16C84	3- 15
AN590	A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs	2- 99
AN591	Apple Desktop Bus	3-169
AN592	Frequency Counter Using PIC16C5X	2- 53
AN593	Serial Port Routines Without Using the RTCC	2-165
AN594	Using the CCP Module	3- 21

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT

Please note that application software written for one family is easily converted to fit another.

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
24CXX serial EEPROM interface	AN515				
93CX6 serial EEPROM interface	AN530				
A/D conversion	AN513	AN546			
AC power line, interface to	AN521				
Addition, 16+16	AN526		AN544		
Addition, fixed point	AN526	AN526	AN544		
Addition, floating point	AN575	AN575	AN575		
Alarm clock implementation	AN529				
Analog to digital conversion	AN513				
Apple desktop bus interface		AN591			
Asynchronous serial port implementation in software	AN510				
Battery management IC				AN570	
Bank switching macros	AN586				
BCD addition	AN526	AN526	AN544		
BCD conversion routines	AN526	AN526	AN544		
BCD subtraction	AN526	AN526			
BCD to binary	AN526	AN526	AN544		
Binary to BCD	AN526	AN526			
Brown-out circuits	AN522	AN522			
Capacitance measurement	AN512				
Capture routines		AN594	AN545		
CCP module		AN594			
Chip select with Vcc					AN535
Clock/calendar implementation	AN529	AN582			
Compare routines		AN594			
Comparison of low-end 8-bit microcontrollers	AN520				
Compatibility - 24C01A					AN517
Compatibility - 93C06					AN516
Data encryption			AN583		
Data Memory Bank Switching Macros	AN586				
D/A using PWM			AN538		
DDE, using PICMASTER's	AN584	AN584	AN584		
DES data encryption			AN583		
Digital voltmeter		AN557			
Digital filters			AN540		
Division, 16/16, unsigned	AN526	AN526	AN544		
Division, 16/16, signed	AN526	AN526	AN544		
Division, fixed point	AN526	AN526	AN544		

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Division, floating point	AN575	AN575	AN575		
DTMF generation			AN543		
EMI Results	AN577				
Endurance					AN537
Endurance predictive model					AN562
FFT			AN575		
Floating point addition	AN575		AN575		
Floating point multiplication	AN575		AN575		
Floating point routines	AN575		AN575		
Floating point subtraction	AN575		AN575		
Frequency Counter	AN592				
Frequency measurement			AN545		
Global interrupts	AN576	AN576	AN576		
I/O expansion			AN547		
I^2C implementation (for serial EE interface only)	AN515			AN571	
I^2C implementaion	AN541	AN554			
I^2C multi-master operation		AN578			
IIR filter			AN540		
Interfacing to 24C32/65					AN558
Interfacing to 24CXX					AN515
Interfacing to 24LCXX					AN567
Interfacing to 93CX6					AN530
Interfacing to 93LC56/66					AN560
Interface to serial A/D	AN531				
Interrupt, disabling global	AN576	AN576	AN576		
Interrupt, PORTB		AN566			
Interrupt, software	AN514				
Keypad interface	AN528/AN529	AN557			
LCD, direct driving with PIC16C5X	AN563				
LCD module interface	AN587	AN587			
LED display interfacing	AN529/590	AN557			
Long calls	AN581				
Low power clock		AN582			
Macros, page and bank switching	AN586				
Math routines	AN526/AN575	AN526/AN575	AN544/AN575		
Measuring capacitance	AN512				
Measuring frequency	AN592		AN545		
Measuring period			AN545		
Measuring pulse-width		AN594	AN545		

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Measuring resistance	AN512				
Measuring temperature	AN512				
Microwire interface	AN531				
Motor control	AN531		AN532		
Mouse, serial	AN519				
MTA11200, calibrating				AN570	
Multiplexing LED and keypad	AN529				
Multiplication, 8 x 8, unsigned	AN526		AN544		
Multiplication, 16 x 16, unsigned	AN526		AN544		
Multiplication, 16 x 16, signed	AN526		AN544		
Multiplication, fixed point	AN526		AN544		
Multiplication, floating point	AN575		AN575		
Oscillator design	AN588	AN588	AN588		
Page switching macros	AN586				
Parallel slave port		AN579			
Parity generation		AN555	AN547		
Period measurement			AN545		
PIC16C84 development programmer			AN589		
PID	AN531		AN532		
PLA implementation	AN511				
PLD replacement using PIC16CXX	AN511				
Pointing device				AN569	
Position control	AN531		AN532		
Positioner	AN531				
Power-on Reset	AN522				
Program Memory Page Switching Macros	AN586				
Pseudo-random number generation			AN544		
Pulse width measurement			AN545		
PWM, using		AN594	AN564		
PWM, choosing frequency		AN594	AN539		
PWM, choosing resolution		AN594	AN539		
PWM generation in software	AN531				
PWM routines		AN594	AN539		
PWM to analog output			AN538		
Quadrature encoder interface			AN532		
Random number generation, Gaussian			AN544		
Random number generation			AN544		
Real time clock (TMR1)		AN580/AN582			

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Real time clock implementation (from AC power line)	AN521				
Real time operating system (RTOS)	AN585	AN585	AN585		
Resistance measurement	AN512				
RS232 interface			AN547		
SEEPROM operation					AN536
Sequencer implementation	AN511				
Serial EEPROM interfacing	AN515				
Serial port (USART) simple routines		AN555	AN547		
Serial A/D interface	AN531				
Servo control			AN532		
Slave port		AN579			
Sine wave generation			AN543		
Software stack	AN527		AN534		
Square root	AN526	AN526	AN544		
Stack management in software	AN527		AN534		
State machine implementation	AN511				
Status save and restore	AN534				
Subtraction, 16 + 16	AN526/AN575	AN575	AN544/AN575		
Subtraction, fixed point					
Subtraction, floating point	AN526		AN544		
Table Read	AN556	AN556	AN548/AN556		
Temperature measurement	AN512				
Thermostat implementation	AN512				
Timer1 in asynchronous mode		AN580			
Tone generation			AN543		
Trajectory generation			AN532		
TrueGauge calibration				AN570	
UART, software implementation	AN510/AN593	AN555			
USART routines			AN547		
Velocity control			AN532		
Voltmeter implementation		AN157			
Wake up on key-stroke	AN528	AN552			
Write cycle optimization					AN559
Zero crossing detect	AN521				

© 1994 Microchip Technology Inc.

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

Motivated by customer requirements....

"Microchip Technology Incorporated draws its impetus from the technology expectations of a large base of long-standing customers. Microchip responds quickly with technology to serve our customers' needs. Moreover, as a fully integrated IC manufacturer, Microchip deploys its panoply of resources to act timely and efficiently, and on a worldwide scale in providing: Technology Development, Design, Wafer Fabrication, Assembly and Test, Quality, Reliability and Customer Support.

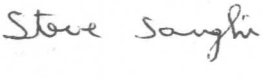
...and powered by continuous improvement...

"Worldwide competition leaves no room for divergence or mediocrity. Microchip Technology, committed to focus on and continuously improve all the aspects of its business, takes pride in its unique corporate culture. To improve performance, our employees are encouraged to analyze their methods continually. Personal empowerment expands the capability of personal responsibility to continually serve our customers better.

...riding, leading and pushing the wave of technological change.

"Our industry's life-line is innovation. The fast pace of technological change is inherent in our industry. Microchip Technology has accelerated the rate of change of its technology and products to leadership in providing field-programmable space-sensitive embedded control solutions.

"Change is our ally. Driving and managing customer-focused change is our winning strategy."



Steve Sanghi
President & Chief Executive Officer

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology Inc.

MICROCHIP TECHNOLOGY INCORPORATED

Company Profile

HIGHLIGHTS

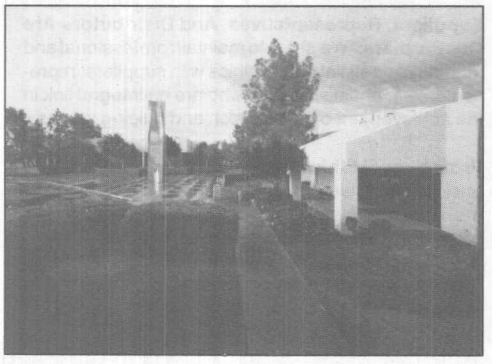
- Focused on providing high-performance, field-programmable embedded control solutions
- An experienced executive team focussed on innovation
- Offers RISC 8-bit field-programmable microcontrollers and supporting logic products
- Offers Serial and Parallel EEPROMs and EPROMs
- Offers complementary Application Specific Standard Products
- Fully integrated manufacturing
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement

BUSINESS SCOPE

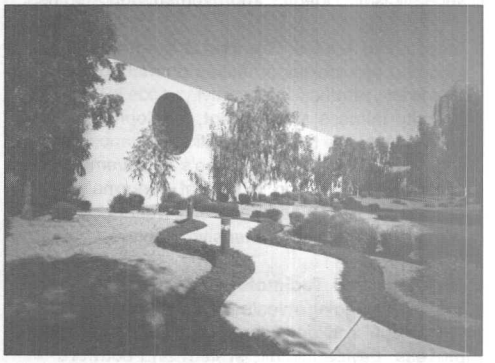
Microchip Technology Incorporated manufactures and markets a variety of VLSI CMOS semiconductor components to support the embedded control market. In particular, the company specializes in highly integrated, RISC microcontrollers, application specific standard products and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in an increasing number of price-sensitive products. Microchip's products feature the industry's most economical OTP (One-Time-Programmable), ROM and EPROM capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

MARKET FOCUS

Microchip Technology targets selected markets where our advanced designs, progressive process technology and industry-leading operating speeds enable us to deliver decidedly superior performance. The company has positioned itself to maintain a dominant role as a supplier of high performance field-programmable microcontrollers and associated memory and logic products for embedded control applications.



Chandler, Arizona:
Company headquarters near Phoenix, Arizona; executive offices, R & D and wafer fabrication occupy this 142,000-square-foot facility.



Tempe, Arizona:
Wafer fabrication capacity is expanded dramatically with the addition of our new 170,000-square-foot facility.

© 1994 Microchip Technology Inc.



MICROCHIP

- Mission Statement -

Microchip Technology Incorporated is a leading supplier of field-programmable embedded control solutions by providing RISC microcontrollers and related non-volatile memory products. In order to contribute to the ongoing success of customers, shareholders and employees, our mission is to focus resources on high value, high quality products and to continuously improve all aspects of our business, providing a competitive return on investment.

- Guiding Values -

Customers Are Our Focus: We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We start by listening to our customers, earning our credibility by producing quality products, delivering comprehensive services and meeting commitments. We believe each employee must effectively serve their internal customers in order for Microchip's external customers to be properly served.

Quality Comes First: We will perform correctly the first time, maintain customer satisfaction and measure our quality against requirements. We practice effective and standardized improvement methods, such as statistical process control to anticipate problems and implement root cause solutions. We believe that when quality comes first, reduced costs follow.

Continuous Improvement Is Essential: We utilize the concept of "Vital Few" to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Employees Are Our Greatest Strength: We design jobs and provide opportunities promoting employee teamwork, productivity, creativity, pride in work, trust, integrity, fairness, involvement, development and empowerment. We base recognition, advancement and compensation on an employee's achievement of excellence in team and individual performance. We provide for employee health and welfare by offering competitive and comprehensive employee benefits.

Products And Technology Are Our Foundation: We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system and software technologies to provide timely, innovative, reliable and cost effective products to support current and future market opportunities.

Total Cycle Times Are Optimized: We focus resources to optimize cycle times to our internal and external customers by empowering employees to achieve efficient cycle times in their area of responsibility. We believe that cycle time reduction is achieved by streamlining processes through the systematic removal of barriers to productivity.

Safety Is Never Compromised: We place our concern for safety of our employees and community at the forefront of our decisions, policies and actions. Each employee is responsible for safety.

Profits And Growth Provide For Everything We Do: We strive to generate and maintain competitive rates of company profits and growth as they allow continued investment for the future, enhanced employee opportunity and represent the overall success of Microchip.

Communication Is Vital: We encourage appropriate, honest, constructive, and ongoing communication in company, customer and community relationships to resolve issues, exchange information and share knowledge.

Suppliers, Representatives, And Distributors Are Our Partners: We strive to maintain professional and mutually beneficial partnerships with suppliers, representatives, and distributors who are an integral link in the achievement of our mission and guiding values.

Professional Ethics Are Practiced: We manage our business and treat customers, employees, shareholders, investors, suppliers, distributors, representatives, community and government in a manner that exemplifies our honesty, ethics and integrity. We recognize our responsibility to the community and are proud to serve as an equal opportunity employer.

© 1994 Microchip Technology Inc.

FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced statistical process control (SPC) and a continuous improvement culture has brought forth tight product consistency levels and high yields which enable Microchip to compete successfully in world markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.

A GLOBAL NETWORK OF PLANTS AND FACILITIES

Microchip is a global competitor providing local service to the world's technology centers. The Company's focal point is the design and technology advancement facility in Chandler, Arizona. Product and technology development is here, along with front-end wafer fabrication and electrical probing.

In late 1993, Microchip purchased a second wafer fabrication facility in Tempe, Arizona–thirteen miles from its existing Chandler, Arizona, operation. The additional 170,000 square foot facility will be equipped with process equipment for use in meeting future production volumes beyond those which could be efficiently produced in Microchip's single existing wafer facility. Initial production from the new Tempe facility is anticipated to begin by late 1994.

Microchip's assembly and test facility in Kaohsiung, Taiwan houses the technology, modern assembly methods and test equipment necessary for plastic and ceramic packaging. Microchip also assembles and tests products in facilities owned and operated by Alphatek in Bangkok, Thailand.

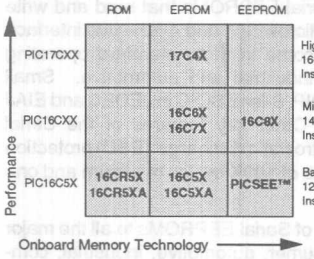
Sales and application offices are located in key cities throughout the Americas, Pacific Rim and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical support, purchasing information and failure analysis.

A PRODUCT FAMILY OF SHARED STRENGTHS

Microchip's product focus is CMOS field-programmable microcontrollers, non-volatile memories and peripherals, and application specific standard products (ASSP). These product lines include PIC16/17 microcontrollers, Serial and Parallel EEPROMs, high-speed EPROMs, and peripherals in a broad range of product densities, speeds and packages.

MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost and small package size. They offer the best price/performance ratio in the industry. Large numbers of these devices are used in automotive and cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.



CMOS PIC16/17 Microcontroller Families

The widely-accepted CMOS PIC16C5X, PIC16CXX and PIC17CXX families are the industry's only 8-bit microcontrollers using a high-speed RISC architecture. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency. The CMOS PIC16C5X family is in high-volume production, shipping in the range of one million units per week, and has achieved more than twenty-five thousand design wins worldwide.

The PIC16CXX mid-range family is rapidly gaining acceptance with three of its members introduced: PIC16C71, PIC16C84 and PIC16C64.

The PIC17CXX family offers the world's fastest execution performance of any 8-bit microcontroller family. The PIC17CXX family extends the PIC16/17 microcontroller's high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. The first member of the family, the PIC17C42, includes a powerful array of intelligent and precise on-chip peripheral features that are ideally suited for many demanding real-time embedded control applications including motor control, process control, security, automotive and medical

© 1994 Microchip Technology Inc.

applications. In addition, the PIC17C42 can function either as a stand-alone microcontroller or can execute instructions from up to 64K words of external program memory. The PIC17C42 features comprehensive timer/counter resources and I/O handling capabilities to address the requirements of complex embedded control applications.

Current CMOS PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded A/D, extended instruction/data memory, inter-processor communication (I²C™ bus, SPI and USARTs) and ROM, RAM, EPROM and EEPROM memories.

Both PIC16CXX and PIC17CXX families are supported by user-friendly development systems including assemblers, software simulators, programmers and emulators.

DEVELOPMENT SYSTEMS

The PICMASTER™ is an advanced real-time in-circuit emulator system running under Windows™ environment. The PICMASTER is a Microchip-designed universal emulator for both PIC16CXX and PIC17CXX families. The PRO MATE™ is an advanced full-featured programmer. PICSTART™ is a low-cost development kit which includes an assembler, simulator and development programmer.

Microchip's Serial EEPROM Designer's Kit includes everything necessary to develop a reliable Serial EEPROM-based design.

SOFTWARE SUPPORT

Both PIC16/17 microcontroller families are supported by assemblers, linker/loaders, libraries and a source-level debugger. The PIC16C5X and PIC16CXX families are also supported by a software simulator.

A full-featured C Compiler and Fuzzy Logic support are also available for all three families.

Customers can obtain on-line updates on Microchip Development Systems and Support Software via the Bulletin Board System (BBS). Please refer to the Microchip BBS product brief in Section 9 for specific access information.

SERIAL EEPROMS

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages. The company has developed the world's first 64K Smart Serial™ EEPROM which currently offers four times the speed, four times the memory and four times the features of any competitive 2-wire Serial EEPROM. Device densities range from 256K bits up to 64K bits. In addition to 5 volt-only operation, Microchip offers Serial EEPROMs that read and write down to 2.5, 2 or 1.8 volts. I²C™, Microwire™ and 4-wire bus interface protocols are standard. Devices come in three standard operating temperature ranges; commercial, industrial and automotive. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths and 14-lead SOIC. Other key features of the Serial EEPROM product line include: electrostatic discharge (ESD) protection greater than 4K volts and endurance of 100K cycles minimum and one million typical.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide, including consumer, automotive, industrial, computer and communications. To date, more than 100 million units have been produced. Microchip is continuing to develop additional unique Serial EEPROMs.

© 1994 Microchip Technology Inc.

PARALLEL EEPROMS

The CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 write and erase cycles typically. Data retention is more than 10 years. Fast write times are less than 200 µsec. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from –40°C to +80°C. Microchip's expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers' needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, pattern recognition and telecommunications.

EPROMS

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. High Speed EPROMs have access times as low as 55 nanoseconds. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in Surface Mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount one-time-programmable (OTP) EPROM market where Microchip is the #1 supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

APPLICATION SPECIFIC STANDARD PRODUCTS (ASSP)

Microchip's Application Specific Standard Product (ASSP) Division provides value-added embedded control solutions by combining PIC16/17 microcontroller architecture with innovative software, silicon and assembly technology. These products incorporate technology that offers a complete solution that is both unique to the customer and standard in manufacture to Microchip. In addition, Microchip ASSPs reduce or remove the barriers for customers to use Microchip solutions in their products through the use of software embedded in secure OTP- or ROM-based microcontrollers. The family is packaged to provide the highest integration to the customer at the best overall system cost.

The MTA11XXX family is the most accurate and most integrated battery management and charging solution available today. The family incorporates Microchip/SPAN patented *TrueGauge*™ technology which digitally integrates battery charge and discharge current to provide an accurate (>97% typical) state of charge indication. The family operates with NiCd, Pb acid and NiMH battery packs from 3 Vdc to 30 Vdc. These products are ideal for portable PC, cellular phone and portable consumer product applications.

Ease of use, low voltage and low cost make the MTA41XXX mouse and trackball MCU firmware solutions ideal for implementing new designs for both PCs and Apple® computers. The products in the MTA41XXX family are 18-lead, low-power CMOS microcontroller ICs combined with application-specific software. By adding a few external components, the user can easily realize a complete mouse or trackball system.

The MTA8XXXX PICSEE™ family of cost-effective system solutions integrate PIC16/17 microcontrollers with EEPROM technology. These PICSEE devices are ideally suited for automotive security, keyless entry, remote control, data acquisition and telecommunication applications. The combined product assembly techniques provide the user the highest performance solution in a compact and cost-effective package.

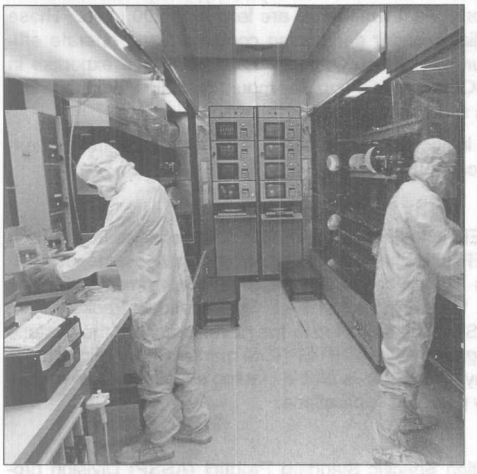
Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

OTHER MICROCHIP PRODUCTS

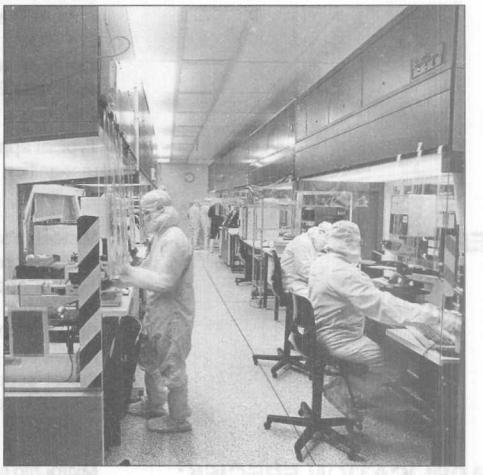
Other Microchip products, such as Liquid Crystal Display (LCD) drivers, are mature products with proven track record and a large, repeat customer base.

© 1994 Microchip Technology Inc.

CHANDLER, AZ FACILITY



Chandler Wafer Fabrication: Diffusion Area



Chandler Wafer Fab: Sub-micron Alignment Area





The Microchip Kaohsiung plant's excellent track record and continuing efforts to achieve higher levels of quality and technological advancement has resulted in superior yields and fast turnaround.

TAIWAN FACILITY

Microchip's assembly and test operation in Kaohsiung, Taiwan received the prestigious Ishikawa Award for assembly and testing excellence.

© 1994 Microchip Technology Inc.

A HISTORY OF INNOVATION

Microchip has a long history of innovation in the semiconductor industry. For more than a quarter century, Microchip and its former parent company have been developers of leading-edge, cost-effective logic and memory products.

Microchip is credited with a number of firsts: The Metal-Oxide-Silicon (MOS) Integrated Circuit, DRAM, Serial EEPROM, Reduced Instruction Set Computer (RISC) microcontroller product family, UART, CMOS 64K EEPROM, and CMOS single chip DSP are all innovations that were originally developed and introduced by Microchip engineers.

FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM and high-speed EPROM products. Advanced process technology modules are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes dimensions down to 0.9 microns.

Microchip's research and development activities, include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools on line. Cycle times for new technology development are continuously reduced by using in-house mask making, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based microcontroller, ASSP and CMOS EEPROM and EPROM products. Objective specifications for new products are developed by listening to our customers and by close cooperation with our many customer-partners worldwide.

QUALITY WITHOUT COMPROMISE

Product reliability is designed into Microchip products at the outset. Wide design margins are established to guarantee that every product can be produced easily, error-free and within the tolerances of the manufacturing process.

All quality assurance tests are tighter than customer specifications. Products are tested at least two machine tolerances tighter than those specified by the customer.

Every new product is qualified under accelerated stress testing. Test samples encompass the full range of processed tolerances at each step. Data sheets detailing these processes enable customers to reach accurate decisions based on known quantitative values.

To determine whether a process is within normal manufacturing variation, industry-leading statistical control techniques are put to work at each process step. In-process controls are performed by operators in the wafer fabrication division and immediate corrective action is taken if they deem a process is out of tight control limits. Products are also sampled weekly through a variety of carefully monitored stress and accelerated life tests.

Microchip's documentation control program assures the correct document is always available at the point of use. Active documents are serialized and stamped to eliminate the possibility of performing a job from obsolete or incorrect instructions.

Individuals in all departments continuously analyze the methods employed at their positions and formulate plans to improve performance. In all areas of our business, everyone is expected to make continuous improvement.

© 1994 Microchip Technology Inc.

A QUALITY AND RELIABILITY ALLIANCE WITH CUSTOMERS

Microchip works together with customers to establish mutual programs to improve the performance of our products in their systems. We go beyond the incoming inspection level and specification by extending our quality and reliability support to the point where the customer ships the system. Microchip's quality programs ensure that our products can be used with such impunity, a customer can implement improvement programs based on Microchip as your leading supplier.

© 1994 Microchip Technology Inc.

SECTION 1
INTRODUCTION TO EMBEDDED SOLUTIONS

Embedded Control Overview 1- 1
PIC16/17 Families Overview and Road Map 1- 1
PIC16C5X: Base-Line Family 1- 1
PIC16CXX: Mid-Range Family 1- 1
PIC17CXX: High-End Family 1- 3
PIC16/17 Naming Convention 1- 4
Serial EEPROM Overview 1- 6
OTP EPROM Overview 1- 6
The Advantages of One-Time-Programmable 1- 6
Application Specific Standard Products 1- 8
Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP™) 1- 8
What's New in the 1994/95 Embedded Control Handbook 1- 10
What's Changed in the 1994/95 Embedded Control Handbook 1- 10

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

Introduction to *The Embedded Control Solutions Company*™

Microchip Technology's mission is to offer leadership semiconductor products for embedded control system applications. To do this we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: 8-bit PIC16/17 microcontrollers and Serial EEPROMS. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

We publish this *Embedded Control Handbook* to assist our customers, existing and new, in their efforts to design and produce state-of-the-art embedded control systems.

EMBEDDED CONTROL OVERVIEW

Unlike "processor" applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are buried inside the application. The user of the product is only concerned with the very top-level user interface (such as keypads, displays and high-level commands). Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in virtually all market segments: consumer, commercial, PC peripherals, automotive, telecommunications (including fast-emerging personal telecom products) and industrial. Most often embedded control products must meet special requirements: cost-effectiveness, low power, small footprint and a high level of system integration.

Typically, most embedded control systems are designed around a microcontroller which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems also require serial EEPROM memories, display drivers, keypads, small displays, etc.

Microchip Technology has established itself as a leading supplier of field-programmable embedded control solutions. The combination of high-performance microcontrollers from both the PIC17CXX and PIC16CXX families, along with industry leading nonvolatile memory products provides the basis for this leadership. Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

PIC16/17 MICROCONTROLLER OVERVIEW AND ROADMAP

Microchip offers three families of 8-bit microcontrollers to best fit your needs:

- PIC16C5X: Base-Line 8-bit Family
- PIC16CXX: Mid-Range 8-bit Family
- PIC17CXX: High-End 8-bit Family

All families offer One-Time-Programmable, low-voltage and low-power options, as well as various packaging options. Selected members are available in ROM version.

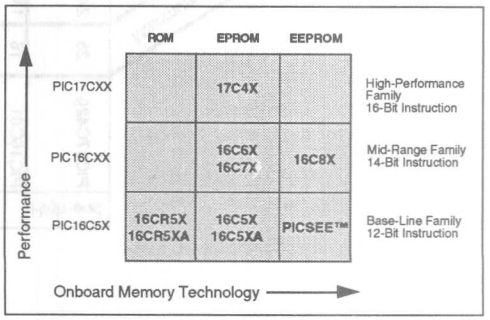
PIC16C5X: BASE-LINE FAMILY

PIC16C5X is the well established base-line family offering the highest cost efficiency. This PIC16C5X products have a 12-bit wide instruction set and are currently offered in 18-, 20- or 28-pin packages. In SOIC and SSOP packaging options, these are the smallest footprint controllers. Low-voltage operation down to 2.0V makes this family ideal for battery operated applications.

PIC16CXX: MID-RANGE FAMILY

PIC16CXX mid-range family offers a wide-range of options, from 18-pin to 44-pin packages as well as low to high level of peripherals integration. This family has a 14-bit wide instruction set. PIC16C71 is a compact microcontroller in a 18-pin package with an 8-bit A/D converter. PIC16C84 offers on-chip EEPROM program and data memory. PIC16C64 is a full-featured 40-pin device with 128 bytes of RAM, three timer/counters, capture, compare, PWM and SPI/I²C™ serial communication port. PIC16C74 is a highly integrated 40-pin device with all the features of the PIC16C64, plus an 8-channel A/D converter and a full-featured USART.

FIGURE 1 - PIC16/17 MICROCONTROLLER MATRIX



© 1994 Microchip Technology Inc.

Introduction

TABLE 1 - FEATURES OVERVIEW OF THE PIC16/17 MICROCONTROLLERS

		Clock	Memory						Peripherals							Features‡		
		Maximum Frequency of Operation (MHz)	Program Memory (words) EPROM	Program Memory (words) ROM	EEPROM	RAM Data Memory (bytes)	EEPROM Data Memory (bytes)	Timer Module(s) †	Capture/Compare/PWM Module	Serial Port (s) (SPI/I²C, SCI)	Parallel Slave Port	Analog to Digital Converter (8-bit)	External Interrupts	Interrupt Sources	I/O*	Voltage Range (Volts)	Number of Instructions	Packages
High-end	PIC17C42§	25	2K	—	—	128	—	TMR0, TMR1, TMR2,TMR3	—	SCI	—	—	Yes	11	33	4.5 - 5.5	55	40-pin DIP, 44-pin PLCC 44 pin QFP
Mid-range	PIC16C61	16	1K	—	—	36	—	TMR0	—	—	—	—	Yes	3	13	3.0 - 6.0	35	18-pin DIP, 18-pin SOIC
	PIC16C64	20	2K	—	—	128	—	TMR0, TMR1, TMR2	1	SPI/I²C™	Yes	—	Yes	8	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C71	16	1K	—	—	36	—	TMR0	—	—	—	4 ch	Yes	4	13	3.0 - 6.0	35	18-pin DIP, 18-pin SOIC
	PIC16C74	20	4K	—	—	192	—	TMR0, TMR1, TMR2	2	SPI/I²C, SCI	Yes	8 ch	Yes	12	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C84	10	—	—	1K	36	64	TMR0	—	—	—	—	Yes	4	13	2.0 - 6.0	35	18 pin DIP, 18 pin SOIC
Base-Line	PIC16C54	20	512	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C54A	20	512	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16CR54	20	—	512	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C55	20	1K	—	—	25	—	RTCC	—	—	—	—	—	—	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C56	20	1K	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C57	20	2K	—	—	72	—	RTCC	—	—	—	—	—	—	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16CR57A	20	—	2K	—	72	—	RTCC	—	—	—	—	—	—	20	2.0 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C58A	20	2K	—	—	73	—	RTCC	—	—	—	—	—	—	12	2.5	33	18-pin DIP, 18-pin SOIC

† PIC17C42 can concatenate Timer1 and Timer2 to form a 16-bit Timer. Timer0 is 16-bit with 8-bit prescaler.
‡ All PIC16/17 Family devices have Power-on Reset, fuse selectable Watchdog Timer and fuse selectable code protect.
§ The PIC17C42 can also operate in microprocessor or external microcontroller mode.
* All PIC16/17 devices offer 20-25mA source / sink current per pin.

© 1994 Microchip Technology

PIC17CXX: HIGH-END FAMILY

The PIC17CXX high-end family currently is comprised of one member, the PIC17C42. Additional members with larger on-chip memories are planned. The PIC17CXX products have a 16-bit wide instruction set. At 25MHz clock rate (160ns instruction cycle). The PIC17C42 offers the highest level of computation power. The PIC17C42 is also highly integrated with peripheral resources.

FIGURE 2 - PIC16/17 MICROCONTROLLER MIGRATION PATH

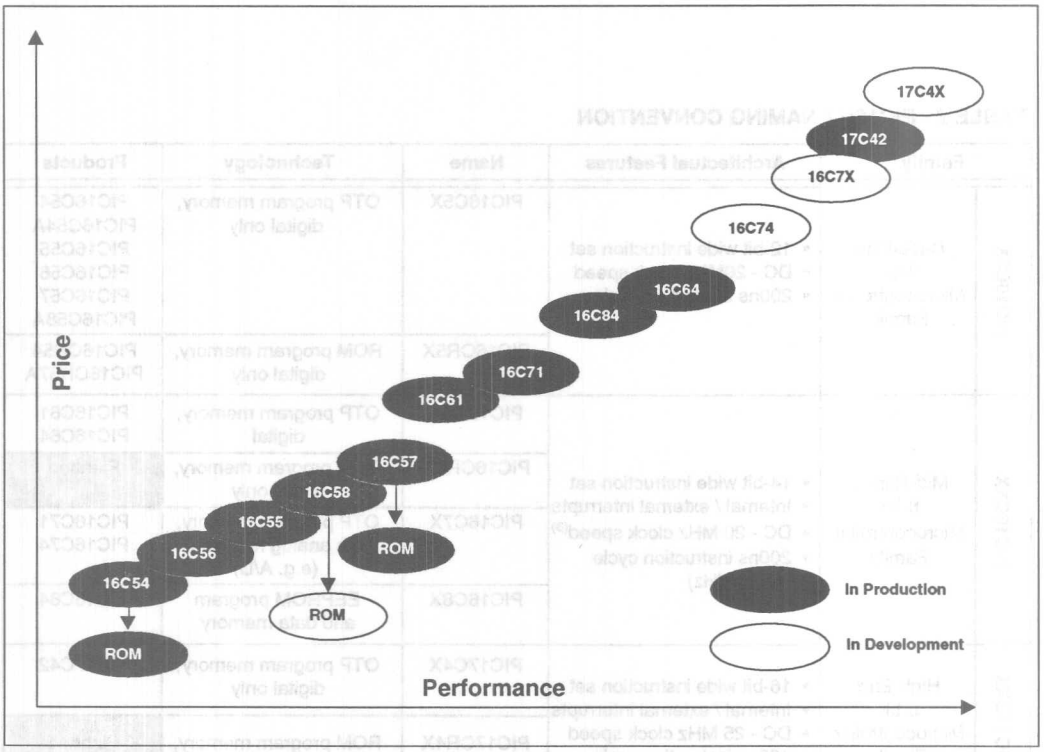


FIGURE 3 - PIC16/17 SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC16CXX	PIC16CXX	PIC17CXX
Assembler	MPASM	✔	✔	✔
Software Simulator	MPSIM	✔	✔	**
C Compiler*	MP-C	✔	✔	✔
Universal Programmer	PRO MATE™	✔	✔	✔
Universal In-Circuit Emulator	PICMASTER™	✔	✔	✔
Fuzzy Logic Development Tool	fuzzyTECH®-MP	✔	✔	✔

* Available from Bytecraft LTD in Canada.
** In development.
For an overview of development tools, see Section 9.

© 1994 Microchip Technology Inc.

PIC16/17 NAMING CONVENTION

The PIC16/17 architecture offers users a wide range of cost/performance options of any 8-bit microcontroller family. In order to identify the families, the following naming conventions have been applied to the PIC16/17 microcontrollers.

TABLE 2 - PIC16/17 NAMING CONVENTION

Family		Architectual Features	Name	Technology	Products
PIC16C5X	Base-Line 8-bit Microcontroller Family	• 12-bit wide instruction set • DC - 20MHz clock speed • 200ns instruction cycle	PIC16C5X	OTP program memory, digital only	PIC16C54 PIC16C54A PIC16C55 PIC16C56 PIC16C57 PIC16C58A
			PIC16CR5X	ROM program memory, digital only	PIC16CR54 PIC16CR57A
PIC16CXX	Mid-Range 8-bit Microcontroller Family	• 14-bit wide instruction set • Internal / external interrupts • DC - 20 MHz clock speed(3) • 200ns instruction cycle (@ 20 MHz)	PIC16C6X	OTP program memory, digital	PIC16C61 PIC16C64
			PIC16CR6X	ROM program memory, digital only	Planned
			PIC16C7X	OTP program memory, with analog functions (e.g. A/D)	PIC16C71 PIC16C74
			PIC16C8X	EEPROM program and data memory	PIC16C84
PIC17CXX	High-End 8-bit Microcontroller Family	• 16-bit wide instruction set • Internal / external interrupts • DC - 25 MHz clock speed • 160ns instruction cycle	PIC17C4X	OTP program memory, digital only	PIC17C42
			PIC17CR4X	ROM program memory, digital only	Planned

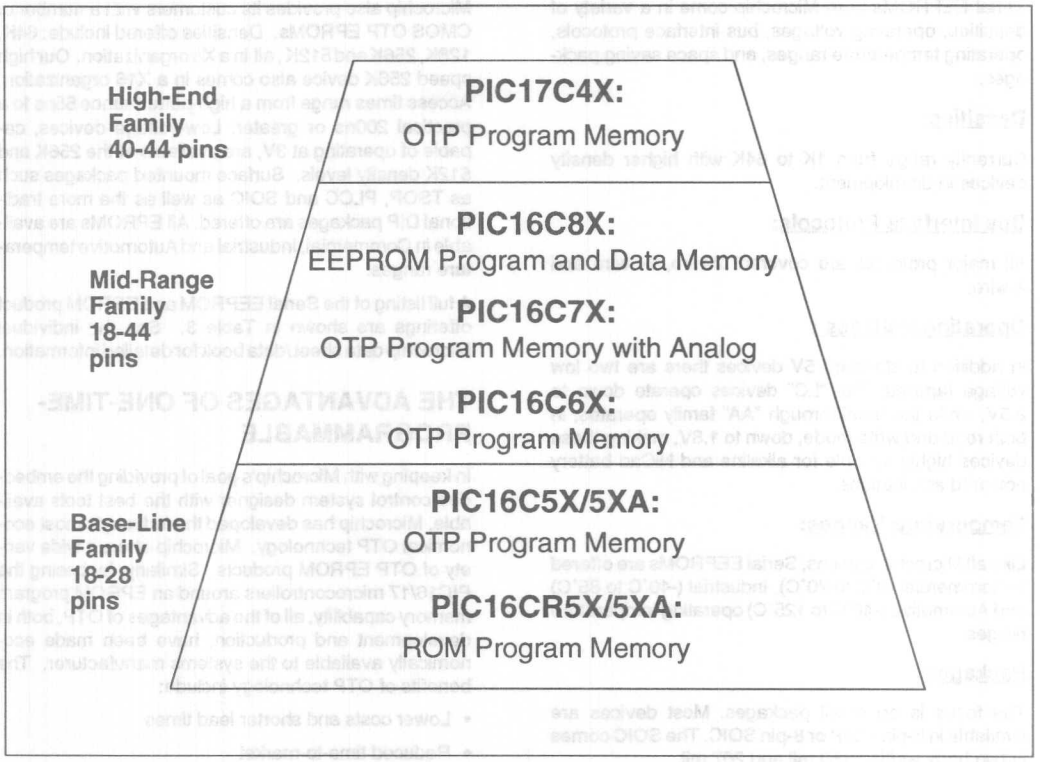
Notes: 1. "A" designates a more advanced process technology, generally offering customers the benefits of lower power, higher speed, etc. (example: PIC16C54, PIC16C54A).

2. The numbering system within each family is not necessarily significant.

3. The maximum clock speed for some devices is less than 20 MHz.

© 1994 Microchip Technology Inc.

FIGURE 4 - PIC16/17 8-BIT MCU FAMILY



SERIAL EEPROM OVERVIEW

Serial EEPROMs from Microchip come in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges, and space saving packages.

Densities:

Currently range from 1K to 64K with higher density devices in development.

Bus Interface Protocols:

All major protocols are covered: 2-wire, 3- wire and 4-wire.

Operating Voltages :

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCad battery powered applications.

Temperature Ranges:

Like all Microchip devices, Serial EEPROMs are offered in Commercial (0°C to 70°C), Industrial (-40°C to 85°C) and Automotive (-40°C to 125°C) operating temperature ranges.

Packages:

The focus is on small packages. Most devices are available in 8-pin PDIP or 8-pin SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Endurance is specified at 1M Erase/Write typical and 100K cycles minimum with a data retention specification greater than 40 years. ESD protection is guaranteed up to 4K volts.

OTP EPROM OVERVIEW

Microchip also provides its customers with a number of CMOS OTP EPROMs. Densities offered include: 64K, 128K, 256K and 512K, all in a X8 organization. Our high speed 256K device also comes in a X16 organization. Access times range from a high-performance 55ns to a practical 200ns or greater. Low-voltage devices, capable of operating at 3V, are available at the 256K and 512K density levels. Surface mounted packages such as TSOP, PLCC and SOIC as well as the more traditional DIP packages are offered. All EPROMs are available in Commercial, Industrial and Automotive temperature ranges.

A full listing of the Serial EEPROM and EPROM product offerings are shown in Table 3. See the individual Microchip data sheet /data book for detailed information.

THE ADVANTAGES OF ONE-TIME-PROGRAMMABLE

In keeping with Microchip's goal of providing the embedded control system designer with the best tools available, Microchip has developed the industry's most economical OTP technology. Microchip offers a wide variety of OTP EPROM products. Similarly, by basing the PIC16/17 microcontrollers around an EPROM program memory capability, all of the advantages of OTP, both in development and production, have been made economically available to the systems manufacturer. The benefits of OTP technology include:

- Lower costs and shorter lead times
- Reduced time-to-market
- In-circuit programming capability
- Code protection via security fuse
- Reduction of inventory requirements at system manufacturing site
- Quick correction of bugs detected in manufacturing
- Quick product feature changes in response to customer requests
- Reduces wasted inventory

© 1994 Microchip Technology Inc.

TABLE 3 - CMOS SERIAL EEPROMS PRODUCT SELECTION GUIDE

FAMILY	Device	Density Organization	Max. Clock Frequency	Endurance (cycles typ.)	Temp. Range	# Pins	Package Types	Operating Voltage
2-Wire (I²C™) Bus Protocol								
STANDARD 2-WIRE FAMILY	24C01A	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C02A	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C04A	4K bits (512 X 8)	100 kHz	1M 100 K	C, I	8 14	P, J SL	5.0V
	85C72	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C82	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C92	4K bits (512 X 8)	100 kHz	1M 100 K	C, I E	8 14	P, J, SM, SL	5.0V
LOW-VOLTAGE 2-WIRE FAMILY	24LC01B	1K bits (128 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC02B	2K bits (256 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC04B	4K bits (512 X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC08B	8K bits (1K X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC16B	16K bits (2K X 8)	400 kHz	1M	C, I	8 14	P, SN SL	2.5V-5.5V
AA LOW-VOLTAGE 2-WIRE FAMILY	24AA01	1K bits (128 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA02	2K bits (256 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA04	4K bits (512 X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA08	8K bits (1K X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA16	16K bits (2K X 8)	100 kHz	1M	C	8 14	P, SN SL	1.8V-5.5V
SMART SERIAL 2-WIRE FAMILY	24C32	32K bits (4K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	4.5V-5.5V
	24LC32	32K bits (4K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	2.5V-6.0V
	24C65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	4.5V-5.5V
	24LC65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	2.5V-6.0V
	24AA65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	1.8V-6.0V
3-Wire (Microwire™)/4-Wire Bus Protocol								
STANDARD 3-WIRE FAMILY	93C06	256 bits (16 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C46	1K bits (64 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C56	2K bits (256 X 8) or (128 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
	93C66	4K bits (512 X 8) or (256 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
LOW-VOLTAGE 3-WIRE FAMILY	93LC46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC46B	1K bits (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56B	2K bits (128 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC66B	4K bits (256 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
AA LOW-VOLTAGE 3-WIRE FAMILY	93AA46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
4-WIRE	59C11	1K bits (128 X 8) or (64 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V

Product Selection as of August 1994

P = Plastic DIP
J = Ceramic DIP
SM = 150mil SOIC
SN = 207mil SOIC
SL = SOIC

C = Commercial (0°C to 70°C)
I = Industrial (–40°C to 85°C)
E = Automotive (–40°C to 125°C)

1

* High Endurance Block.

© 1994 Microchip Technology Inc.

APPLICATION SPECIFIC STANDARD PRODUCTS

The Application Specific Standard Products (ASSP) Division complements and strengthens Microchip's leadership position in 8-bit microcontrollers and related specialty memory products for the embedded control market. The ASSP Division employs innovative multi-chip module packaging, applications expertise, firmware and new technology to create integrated, single-chip solutions for specific high-volume embedded control applications such as PC pointing devices, TrueGauge™ battery management, and PICSEE™ microcontrollers. By offering more complete solutions, Microchip can provide its customers with higher value-added products, with the additional benefits of faster time-to-market and lower design overhead. A full ASSP product listing is shown in Table 4.

EASE OF PRODUCTION UTILIZING QUICK TURN PROGRAMMING (QTP) AND SERIALIZED QUICK TURN PROGRAMMING (SQTP)

Recognizing the needs of high-volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick Turn Programming allows factory programming of OTP product prior to delivery to the system manufacturing operation. PIC16/17, EPROM and Serial EEPROM products can be automatically programmed with the users program during the final stages of the test operation at Microchip's assembly and test operations in Philippine Islands, Taiwan and Thailand. This low-cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PIC16/17 and memory products as custom ROM products. With one- to four-week lead times on QTP product, the user no longer needs to plan for the extended ROM masking lead times and masking charges associated with custom ROM products. This capability, combined with the off-the-shelf availability of standard OTP product, ensures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique in the 8-bit microcontroller market is Microchip's ability to enhance the QTP capability with Serialized Quick Turn Programming (SQTP). SQTP allows for the programming of devices with unique, random or serialized identification codes. As each PIC16/17 device is programmed with the customers program code, a portion of the program memory space can be programmed with a unique code, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network are essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or nonvolatile memory components. The SQTP offering, pioneered by Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

© 1994 Microchip Technology Inc.

TABLE 4 - ASSP PRODUCT SELECTION GUIDE

FIRMWARE PRODUCTS	Device	Interface	Feature	Operating Voltage	Temp. Range	Number of Pins	Package Types
BATTERY MANAGE-MENT	MTA11200 TrueGauge™	1-wire and RS232	Monitor and Charge Control for NiCD, NiMH Lead Acid	3.0V-6.25V	C,I	28	PDIP, SPDIP SOIC, SSOP
POINTING DEVICES	MTA41300	Serial, PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41120	ADB	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41110	PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41111	PS/2	Velocity Scaling Trackball Controller	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP

PICSEE PRODUCTS						
Device	Speed	Feature	Operating Voltage	Temp. Range	Number of Pins	Package Types
MTA81010-RC	DC to 4 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC, JW
MTA81010-XT	DC to 4 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC, JW
MTA8R1010-RC	DC to 4 MHz	512 x 12 ROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA8R1010-XT	DC to 4 MHz	512 x 12 ROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA81010-LP	DC to 40 KHz	512 x 12 EPROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA8R1010-LP	DC to 40 KHz	512 x 12 ROM	2.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA85401	DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85411	DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85402	DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85412	DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85801	DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85811	DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85802	DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85812	DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP

© 1994 Microchip Technology Inc.

WHAT'S NEW IN MICROCHIP'S 1994/95 EMBEDDED CONTROL HANDBOOK

Organization

The book is reorganized for increasing complexity of Application Notes in each section. A new section for generic PIC16/17 Application Notes has been added.

New Application Notes

SECTION 2 - PIC16C5X

Implementing Long Calls (AN581)
Macros for Page and Bank Switching (AN586)
Frequency Counter Using PIC16C5X (AN592)
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs (AN590)
Serial Port Routines Without Using the RTCC (AN593)
PIC16C54A EMI Results (AN577)

SECTION 3 - PIC16CXX

Using the CCP Module (AN594)
Interfacing to an LCD Module (AN587)
Using the 8-Bit Parallel Slave Port (AN579)
Using Timer1 in Sleep (AN580)
Low Power Clock (AN582)
Apple Desktop Bus (AN591)
I^2C Multi-Master Mode (AN578)
Programming the PIC16C84 Microcontroller (AN589)

SECTION 4 - PIC17CXX

Implementation of the Data Encryption Standard Using PIC17C42 (AN583)

SECTION 5 - PIC16/17

Disabling Global Interrupts (AN576)
IEEE 754 Compliant Floating Point Routines (AN575)
Real Time Operating System (AN585)
PIC16/17 Oscillator Design Guide (AN588)
Using PICMASTER's DDE (AN584)

SECTION 6 - ASSP

Calibrating the MTA11200 (AN570)
Communicating with EEPROM in MTA85XXX (AN571)
Hardware and Software Resolution for a Pointing Device (AN569)

SECTION 8 - SERIAL EEPROMS

Questions and Answers Concerning Serial EEPROMs (AN572)

SECTION 9 - DEVELOPMENT TOOLS

System Information Hotline
PICDEM-2
MP-C
*fuzzy*TECH-MP

WHAT'S CHANGED IN MICROCHIP'S 1994/95 EMBEDDED CONTROL HANDBOOK

Modified / Corrected Application Notes

Many application notes have had many "minor" corrections implemented, such as spelling and source code alignment. The following list is intended to point out the application notes with "major" modifications or corrections.

SECTION 3 - PIC16CXX

Software Implementation of Asynchronous Serial I/O (AN555)

SECTION 4 - PIC17CXX

PIC17CXX Math Utility Routines (AN544)

SECTION 5 - PIC16/17

PIC16C5X / 16CXX Math Utility Routines (AN526)

SECTION 8 - SERIAL EEPROMS

Interfacing 24LCXXB Serial EEPROMS to the PIC16C54 (AN567)
Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code (AN558)
Using the 93LC56 and 93LC66 (AN560)

SECTION 9 - DEVELOPMENT TOOLS

The Development Tools Section has been updated.

© 1994 Microchip Technology Inc.

SECTION 2
PIC16C5X APPLICATION NOTES

A Comparison of Low End 8-Bit Microcontrollers - AN520 ..... 2- 1
Power-Up Considerations - AN522 ..... 2- 11
Software Interrupt Techniques - AN514 ..... 2- 15
Software Stack Management - AN527 ..... 2- 19
Implementing Long Calls - AN581 ..... 2- 23
Macros for Page and Bank Switching - AN586 ..... 2- 27
Implementing Wake-Up on Keystroke - AN528 ..... 2- 47
Interfacing to AC Power Lines - AN521 ..... 2- 51
Frequency Counter Using PIC16C5X - AN592 ..... 2- 53
Analog to Digital Conversion - AN513 ..... 2- 59
Implementing Ohmeter/Temperature Sensor - AN512 ..... 2- 65
Implementing a Simple Serial Mouse Controller - AN519 ..... 2- 71
Intelligent Remote Positioner - AN531 ..... 2- 83
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs - AN590 ..... 2- 99
Multiplexing LED Drive and a 4 x 4 Keypad Sampling - AN529 2-109
Using PIC16C5X Microcontrollers as LCD Drivers - AN563 2-133
PLD Replacement - AN511 2-145
Serial Port Routines Without Using the RTCC - AN593 2-165
Implementation of an Asynchronous Serial I/O - AN510 2-171
Using PIC16C5X as a Smart I²C™ Peripheral - AN541 2-193
PIC16C54A EMI Results - AN577 2-207

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

MICROCHIP

AN520

A Comparison of Low End 8-Bit Microcontrollers

INTRODUCTION

The PIC16C5X Family of microcontrollers from Microchip Technology, Inc. provides significant execution speed and code-compaction improvement over any other 8-bit microcontroller in its price range.

The superior performance of the PIC16C5X microcontrollers can be attributed primarily to its RISC-like architecture. The PIC16C5X employs Harvard architecture, i.e., has separate program memory space (12-bit wide instructions) and data memory space (8-bit wide data). It also uses a two stage pipelining instruction fetch and execution. All instructions are executed in a single cycle (200 ns @ 20 MHz clock) except for program branches which take two cycles, and there are only 33 instructions to remember.

Separation of program and data space allows the instruction word to be optimized to any size (12-bit wide in case of PIC16C5X). This makes it possible, for example, to load an 8-bit immediate value in one cycle. First, because there is no conflict between instruction fetch and data fetch (as opposed to von Neumann architecture) and secondary because the instruction word is wide enough to hold the 8-bit data.

In the following sections we will compare the PIC16C5X @ 20 MHz with:

- SGS-Thomson ST62 @ 8 MHz
- Motorola MC68HC05 @ 4.2 MHz
- Intel 8048/8049 @ 11 MHz
- Zilog Z86CXX @ 12 MHz
- National COP800 @ 20 MHz

Several coding examples will be considered. While the comparisons are not entirely scientific, they will, nevertheless, demonstrate to the reader the relative superior performance of the PIC16C5X. The examples chosen here are used frequently in microcontroller applications.

PACKING BCD

This example will take two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

PIC16C5X

		Byte/Words	Cycles
SWAPF	REGHI,W	1	1
IORWF	REGLO	1	1
		2	2

0.4 µs

COP800

		Byte/Words	Cycles
X	A,[B+]	1	2
SWAP	A	1	1
OR	A,[B]	1	1
X	A,[B]	1	1
		4	5

B is pointing to the higher BCD digit initially. After auto-increment, it points to the lower BCD digit.

5 µs

ST62

		Byte/Words	Cycles
LD	A, REGHI	2	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
ADD	A, REGLO	1	4
LD	REGLO, A	2	4
		10	28

REGHI & REGLO are registers addressable by short direct addressing mode.

45.5µs

MC68HC05

		Byte/Words	Cycles
LDA	REGHI	2	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ADD	REGLO	2	3
STA	REGLO	2	4
		10	22

10.5 µs

Z86CXX

		Byte/Words	Cycles
SWAP	REGHI	2	8
OR	REGHI,REGLO	2	6
		4	14

REGHI and REGLO are addressable via the working register addressing mode.

5.33 µs

8048/8049

		Byte/Words	Cycles
MOV	A,Rx	1	1
SWAP	A	1	1
ORL	A,Ry	1	1
MOV	Ry,A	1	1
		4	4

Register Rx contains higher BCD digit, Ry holds lower BCD digit.

5.45 µs

© 1994 Microchip Technology Inc.

LOOP CONTROL

This example is one of simple loop control where a register containing loop count is decremented, tested for zero and if not branched back to the beginning of the loop.

PIC16C5X

		Byte/Words	Cycles
DECFSZ	COUNT	1	1/2
GOTO	BEG_LOOP	1	2/-
		2	3/2

0.6µs/0.4 µs

COP800

		Byte/Words	Cycles
DRSZ	COUNT	1	3
JP	BEG_LOOP	1	3
		2	6

COUNT is Register (RAM F0h-FFh). 6 µs

ST62

		Byte/Words	Cycles
DEC	X	1	4
JRZ	BEG_LOOP	1	2
		2	6

9.75 µs

MC68HC05

		Byte/Words	Cycles
DECX		1	3
BEQ	BEG_LOOP	2	3
		3	6

2.86 µs

Z86CXX

		Byte/Words	Cycles
DJNZ	COUNT, BEG_LOOP	2	10/12

1.67 µs/2 µs

8048

		Byte/Words	Cycles
DJNZ	Rx, BEG_LOOP	2	2

2.73 µs

Bit Test & Branch

This example tests a single bit in a register or a RAM location and makes a conditional branch. We assume that MSB is tested and a branch is to be taken if the bit is set.

PIC16C5X

		Byte/Words	Cycles
BTFSC	REG, 7	1	1/2
GOTO	NEWADD	1	2/-
		2	3/2

0.6 µs/0.4 µs

COP800

		Byte/Words	Cycles
IFBIT	7, [B]	1	1
JP	NEWADD	1	3
		2	4

B points to the memory location under test. 4 µs

ST62

		Byte/Words	Cycles
JRR	7, NEWADD	3	5

8.125 µs

MC68HC05

		Byte/Words	Cycles
BRCLR	7, NEWADD	3	5

2.38 µs

Z86CXX

		Byte/Words	Cycles
BTJRT	NEWADD, REG, 7	3	16/18

2.67 µs/3.0 µs

8048/8049

		Byte/Words	Cycles
MOV	A, @Rx	1	1
ANL	A, #80H	2	2
JNZ	NEWADD	2	2
		5	5

Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs

© 1994 Microchip Technology Inc.

Shifting Out 8-Bit Data & Clock

We will now consider the task of serially shifting out an 8-bit data and clock. Data and clock outputs are generated under program control by toggling two output pins.

Data is transmitted on the rising edge of the clock. No attempt is made to make the clock output symmetrical in order to make the code efficient. Data out is guaranteed on the falling edge of the clock. These conditions are satisfactory for most applications.

PIC16C5X

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	MOVLW	08H	; Bit Count	1	1	1	
	MOVWF	BITCNT	;	1	1	1	
XM1	BCF	PORTB,0	; 0 → Data Out Pin	1	1	1	
	BCF	PORTB,1	; 0 → Clock Out Pin	1	1	1	
	RRF	XDATA	; Rotate Right thru Carry	1	1	1	
	BTFSC	STATUS,CARRY	; Test Carry Bit	1	2	1	
	BSF	PORTB,0	; 1→ Data Out Pin	1	–	1	Loop
	BSF	PORTB,1	; 1→ Clock Out Pin	1	1	1	
	DECFSZ	BITCNT	; Decrement Count ; Skip if Zero	1	1	1	
	GOTO	XM1	;	1	2	2	
	BCF	PORTC,1	; 0→ Clock	1	1	1	
				11	74	74	

Transmit time is the same for 00h or FFh: 74 Tcyc = 14.8 µs. Note that there was no need to load the data in the Accumulator (W) since the PIC16C5X can operate directing on file registers.

COP800

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LD	A,XDATA	; Load Data in Acc.	2	3	3	
	LD	BITCNT #08H	; Load Bit Count	2	3	3	
	LD	B,#D0H	; B Points to PORTL	2	3	3	
XM1	RBIT	0,[B]	; 0 → Clock	1	1	1	
	RBIT	1,[B]	; 0 → Data	1	1	1	
	RRCA		; Rotate A Right thru Carry	1	1	1	
	IFC		;	1	1	1	Loop
	SBIT	1,[B]	; 1 → Data	1	–	1	
	SBIT	0,[B]	; 0 → Clock	1	1	1	
	DRSZ	BITCNT	; Decrement Bit Count	1	3	3	
	JP	XM1	; and Go Back if ≠ 0	1	3	3	
	RBIT	0,[B]	;	1	3	3	
				16	100	108	

Accumulator A is first loaded with the data word. Transmit time is maximum for data = FFh: 105 Tcyc = 105 µs.

ST62

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
	LDI	A, #08		2	4	4	
	LD	X, A	; Bit Count	1	4	4	
	LD	A, W	;Xmit Data	1	4	4	
XM1	RES	0, DRB	; 0 → Clock	2	4	4	
	RES	1, DRB	; 0 → Data	2	4	4	
	SLA	A	;	2	4	4	
	JRNC	XM2	;	1	2	2	
XM2	SET	1, DRB	; 1 → Data	2	–	4	Loop
	SET	0, DRB	; 1 → CLK	2	4	4	
	DEC	X	;	1	4	4	
	JRNZ	XM1	;	1	2	2	
	RES	0, DRB	; 0→ Data	2	4	4	
				19	208	240	

Register W contains the Data word.
Transmit time for FFh = 240 cycles = 390 µs.

© 1994 Microchip Technology Inc.

SHIFTING OUT 8-BIT DATA AND CLOCK (CONT.)

MC68HC05

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LDA	XDATA	; Load Xmit Data	2	3	3	
	LDX	#$08	; Load Bit Count	2	2	2	
XM1	BCLR,	0,PORTB	; 0 → Clock	2	5	5	Loop
	BCLR	1,PORTB	; 0 → Data	2	5	5	
	ROLA		;	1	3	3	
	BCC	XM2	;	2	3	3	
	BSET	1,PORTB	; 1 → Data	2	–	5	
	BSET	0,PORTB	; 1 → Clock	2	5	5	
XM2	DECX		;	1	3	3	
	BNE	XM1	;	2	3	3	
	BCLR	0,PORTB	; 0 → Data	2	5	5	
				20	226	266	

Transmit time is maximum for transmitting FFh = 266 cycles = 126.7 µs.

Z86CXX

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LD	COUNT,#8	; Load Bit Count	3	10	10	
	AND	P2,#%FC	; 0 → Data, Clock	2	6	6	
XM1	RRC	XDATA	;				
	JR	NC,XM2	;				
	OR	P2, #01	; 1 → Data	2	6	6	Loop
				2	12	10	
XM2	OR	P2, #02	; 1 → Clock	3	–	10	
	DJNZ	COUNT,XM1	;				
	AND	P2,#%FC	; 0 → Clock, Data	3	10	10	
				2	12	12	
				3	10	10	
				21	348	412	

Transmit time is maximum for transmitting FFh = 412 cycles = 68.67 µs.

8048/8049

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	MOV	A,@R0	; R0 Points to Data Word	1	1	1	
	MOV	R1,#08H	; Load Bit Count	2	2	2	
XM1	ANL	PORT1,#0FCH	; 0→ Data, Clock	2	2	2	Loop
	RRC	A	; Rotate Right A thru Carry	1	1	1	
	JC	XM2	;	2	2	2	
	ORL	PORT1,#01H	; 1 → Data	2	–	2	
XM2	ORL	PORT1,#02H	; 1→ Clock	2	2	2	
	DJNZ	R1,XM1	; Decrement Count	2	2	2	
				14	75	91	

Transmit time is maximum for transmitting FFh = 91 cycles = 124.1 µs.

© 1994 Microchip Technology Inc.

Software Timer

Microcontrollers quite often need to implement time delays. Debouncing key input, pulse width modulation, and phase angle control are just a few examples. Implementing a 10 ms time delay loop subroutine will be considered in this section.

PIC16C5X

```
                                                        Byte/Words   Cycles
DELAY   MOVLW   41H         ;10 ms Delay Loop               1          1
        MOVWF   COUNT2      ;                               1          1
        CLRF    COUNT1      ;                               1          1

LOOP    INCFSZ  COUNT1      ;This Inner Loop will be        1          2/1
        GOTO    LOOP        ; Executed 256 Times            1          2
        DECFSZ  COUNT2      ;                               1          2/1
        GOTO    LOOP        ;                               1          2
        RET                 ;                               1          2
                                                            8
```

Execution time for the routine = 5 + (255 x 3 + 5) • 65 = 20025 Tcyc = 10.011 ms. The PIC16C5X can implement delay times very precisely (when necessary) because of its fine instruction cycle resolution.

COP800

```
                                                        Byte/Words   Cycles
DELAY   LD      COUNT1,#0BH ;10 ms Delay Loop               2          3
        LD      B,#0EH      ;                               1          1

LOOP    DRSZ    B           ;                               1          1
        JP      LOOP        ;                               1          1
        DRSZ    COUNT1      ;                               1          1
        JP      LOOP        ;                               1          1
        RET                 ;                               1          5
                                                            8
```

Execution time for the routine = (6N2 + 6) N1 + 9 cycles. Here N1 = 0BH and N2 = 0EH, which gives us: 999 Tcyc = 9.99 ms.

ST62

```
                                                        Byte/Words   Cycles
        LDI     A, #FF                                      2          4
        LD      X, A        ; LOOP1 Count                   1          4
        LDI     A, #04                                      2          4
        LD      Y, A        ; LOOP2 Count                   1          4
LOOP    DEC     X           ; 0      CLK                    1          4
        JRNZ    LOOP        ;                               1          2
        DEC     Y           ; 0      CLK                    1          4
        JRNZ    LOOP                                        1          2
                                                           10
```

Execution time for the subroutine = (6N1 + 6) N2 + 16 cycles, where N1 = FFh, N2 = 04 gives us 10.01 ms.

MC68HC05

```
                                                        Byte/Words   Cycles
DELAY   LDX     $2D         ;10 ms Delay Loop               2          2
        LDX     $5C         ;                               2          2

LOOP    DECA                ;                               1          3
        BNE                 ;                               2          2
        DECX        LOOP    ;                               1          3
        BNE                 ;                               2          2
        RTS         LOOP    ;                               1          6
                                                           11
```

Execution time for the subroutine = (5 x N1 + 5) N2 + 10, with N1 = 2DH, N2 = 5CH, time delay = 10.081 ms.

© 1994 Microchip Technology Inc.

SOFTWARE TIMER (CONT.)

Z86CXX

```
                                                          Byte/Words   Cycles
DELAY   LD      COUNT1,#%61     ;10 ms Delay Loop             2          6
        LD      COUNT2,#%33     ;                             2          6

LOOP    DJNZ    COUNT1,LOOP     ;
        DJNZ    COUNT2,LOOP     ;                             2        10/12
        RET                     ;                             2        10/12
                                                              1          14
                                                              9
```

Total execution time = (12N1 + 10) N2, with N1 = 61H, N2 = 33H, time delay = 59976 cycles = 9.979 ms.

8048/8049

```
                                                          Byte/Words   Cycles
DELAY   MOV     COUNT1,#13H     ;10 ms Delay Loop             2          2

LOOP1   MOV     COUNT2,#AFH     ;                             2          2

LOOP2   DJNZ    COUNT2,LOOP2    ;                             2          2
        DJNZ    COUNT1,LOOP1    ;                             2          2
        RET                     ;                             1          2
                                                              9
```

Execution time for the subroutine = (2N1 + 4) N2 + 4 cycles. Here N1 = 13H, N2 = AFH, which gives us: 7354 cycles = 10.028 ms.

© 1994 Microchip Technology Inc.

SUMMARY

Table 1 summarizes code sizes for different microcontrollers. The overall relative code size number is an average of the individual relative code sizes. Given that the PIC16C5X's program word size is 12-bit, whereas all the other microcontrollers have 8-bit program memory, a compaction of 1.5 µs is expected. Clearly, the PIC16C5X meets this compaction (except for the COP800) and exceeds in most comparisons.

Table 2 summarizes relative execution speed. The overall speed is an average of relative speed numbers. For example, the COP800 will, on an average, exhibit 27% of the code execution speed of a PIC16C5X. In other words, the PIC16C5X will be 1/0.27 = 3.7 times faster than a COP800 on an average.

TABLE 1 - COMPARISON OF CODE EFFICIENCY*

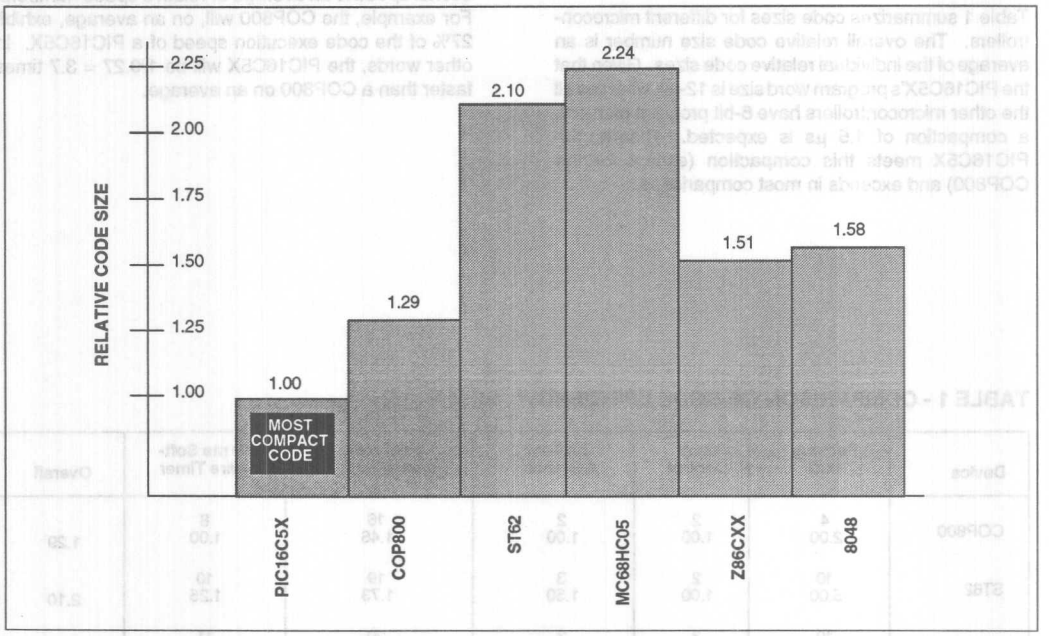
Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800	4 2.00	2 1.00	2 1.00	16 1.46	8 1.00	1.29
ST62	10 5.00	2 1.00	3 1.50	19 1.73	10 1.25	2.10
MC68HC05	10 5.00	3 1.50	3 1.50	20 1.82	11 1.38	2.24
Z86CXX	4 2.00	2 1.00	3 1.50	21 1.91	9 1.125	1.51
8048/8049	4 2.00	2 1.00	5 2.51	14 1.28	9 1.13	1.58
PIC16C5X @ 8 MHz	2	2	2	11	8	1.00

* In each box, the top number is the number of program memory locations required to code the application. The bottom number is relative code size compared to the PIC16C5X:

$$\frac{\text{\# program memory locations for other microcontroller}}{\text{\# program memory locations for the PIC16C5X}}$$

© 1994 Microchip Technology Inc.

FIGURE 1 - CODE SIZE COMPARISON



© 1994 Microchip Technology Inc.

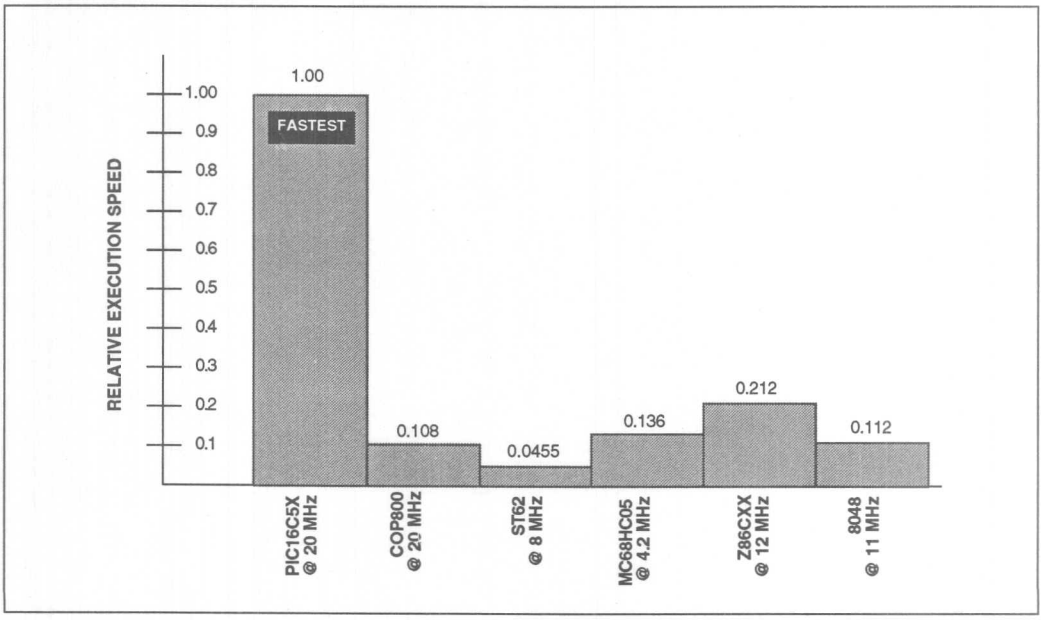
TABLE 2 - COMPARISON OF EXECUTION SPEED*

Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800 @ 20 MHz	5 µs 0.08	6 µs 0.0832	4 µs 0.1252	105 µs 0.1408	–	0.108
ST62 @ 8 MHz	45.5 µs 0.0088	9.75 µs 0.0615	8.125 µs 0.0738	390 µs 0.0329	–	0.0455
MC68HC05 @ 4.2 MHz	10.05 µs 0.038	2.86 µs 0.1748	2.38 µs 0.21	126.7 µs 0.1168	–	0.136
Z86CXX @ 12 MHz	2.33 µs 0.172	1.835 µs 0.272	2.835 µs 0.176	68.67 µs 0.224	–	0.212
8048/8049 @ 11 Mhz	5.45 µs 0.0732	2.73 µs 0.1824	6.82 µs 0.0732	124.1 µs 0.1196	–	0.112
PIC16C5X @ 20 MHz	0.4 µs	0.6/0.4 µs	0.6/0.4 µs	14.8 µs	–	1.00

* In each box, the top number is the time required to execute the example code, while the bottom number is a measure of relative performance compared to the PIC16C5X:

$$\frac{\text{time required to execute code by the PIC16C5X}}{\text{time required to execute code by other microcontroller}}$$

FIGURE 2 - EXECUTION SPEED COMPARISON



© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

AN522

Power-Up Considerations

INTRODUCTION

When powering up all microcontrollers it is necessary for the power supply voltage to traverse voltage ranges where the device is not guaranteed to operate before the power supply voltage reaches its final state. Since some circuits on the device (logic) will start operating at voltage levels lower than other circuits on the chip (memory), the device may power-up in an unknown state. To guarantee that the device starts up in a known state, it is necessary that it contain a power-up reset circuit. PIC16C5X microcontrollers are equipped with on-chip power-on reset circuitry, which eliminates the need for external reset logic. This circuit will function in most power-up situations where VCC rise time is fast enough (50 ms or less). This application note describes the typical power-up sequence for PIC16C5X microcontrollers. Methods of assuring reset on power-up and after a brownout are discussed and simple, low cost external solutions are discussed for power-up situations where the PIC16C5X's internal circuitry cannot provide the reset.

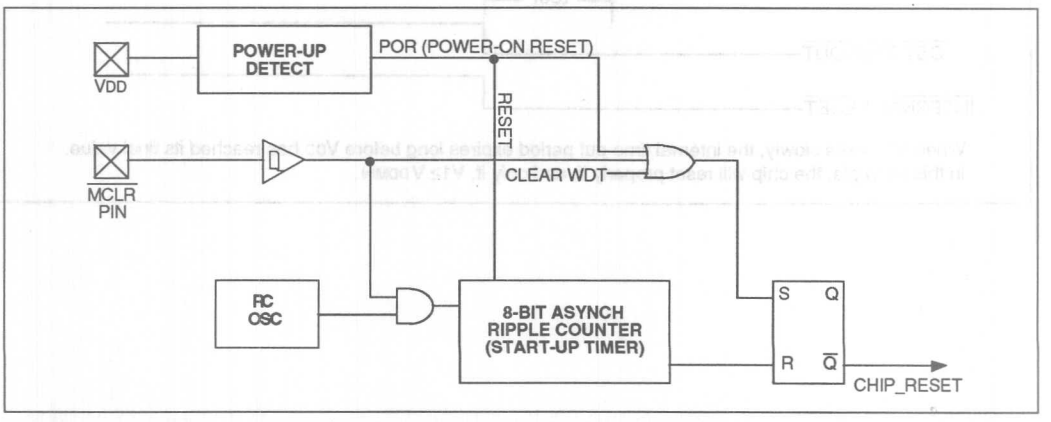
POWER-UP SEQUENCE

The PIC16C5X incorporates complex power-on reset (POR) circuitry on-chip which provides solid, reliable internal chip reset for most power-up situations. To use this feature, the user merely needs to tie MCLR to VDD. A simplified block diagram of the on-chip reset circuitry is shown in Figure 1. On power-up, the reset latch and the start-up timer are reset to appropriate states by the power-on reset (POR). The start-up timer will begin counting once it detects MCLR to be high (i.e., external chip reset goes inactive). After the time-out period, which is typically 18 ms long, the timer will reset the reset latch and thus end the on-chip reset signal.

Figures 2 and 3 are two power-up situations with relative fast rise time on VDD. In Figure 1, VDD is stable when MCLR is brought high (i.e., reset pulse is being provided by external source). The chip actually comes out reset about tOST ms after that, where tOST = oscillator start-up timer. (The timer is called oscillator start-up timer because the time-out was incorporated primarily to allow the crystal oscillator to stabilize on power-up.) In Figure 3, the MCLR and VDD are tied together and clearly the on-chip rest mechanism is being utilized. The VDD is stable before the start-up timer expires and there is no problem with proper reset.

Figure 4, where VDD rise time is much greater than tOST (typically 18 ms) clearly is the potentially problematic situation. The POR (power-on reset) pulse comes when VDD is about 1.5V. Most CMOS logic, including the start-up timer starts functioning between 1.5V to 2.0V. When the start-up timer starts times out, the chip reset is ended and the chip attempts to execute. If by this time the VDD has reached VDD MIN value, then all circuits are guaranteed to function correctly and power-up reset is successful. If, however, the VDD slope was too slow and had not reached VDD MIN, then the chip may or may not function properly.

FIGURE 1 - PIC16C5X INTERNAL RESET CIRCUIT



© 1993 Microchip Technology Inc.

FIGURE 2 - EXTERNAL RESET PULSE

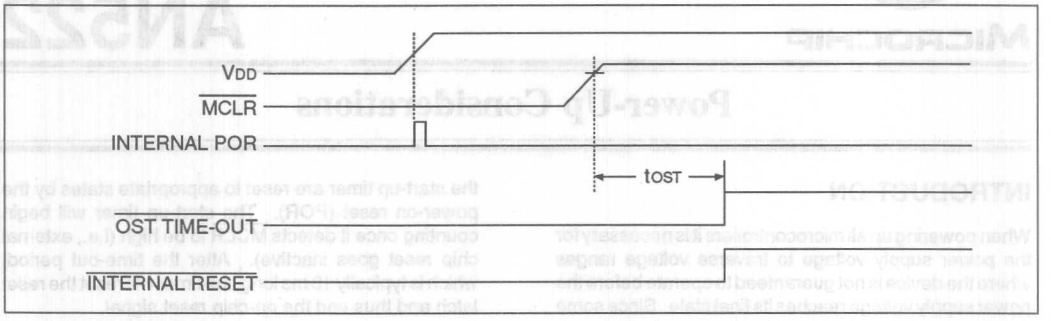


FIGURE 3 - INTERNAL RESET (VDD AND MCLR TIED TOGETHER)

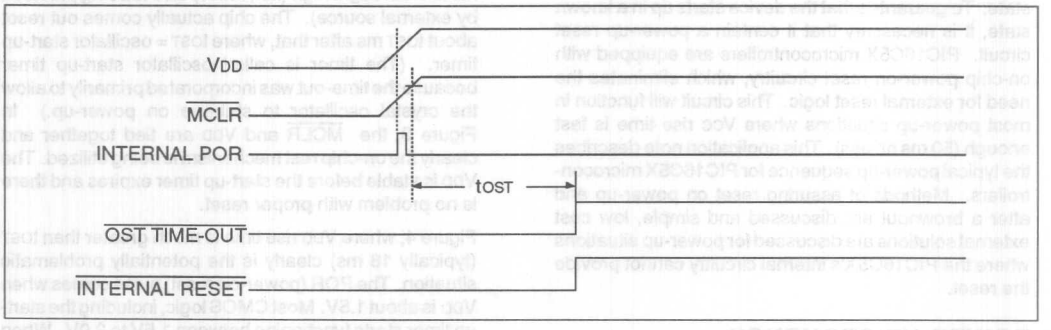
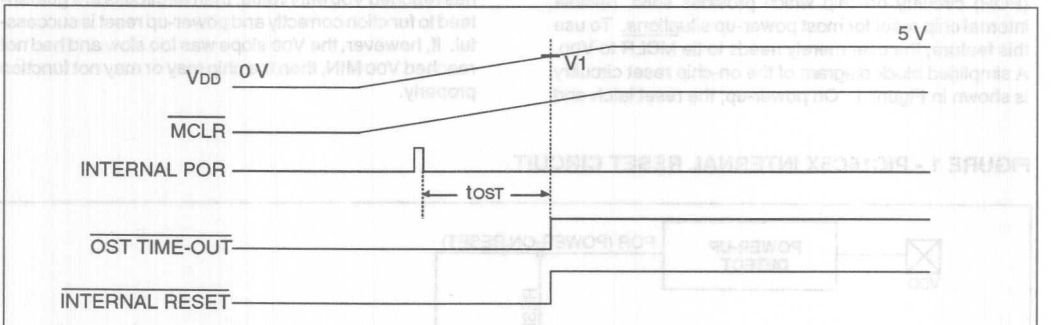


FIGURE 4 - INTERNAL RESET (VDD AND MCLR TIED TOGETHER): SLOW VDD RISE TIME



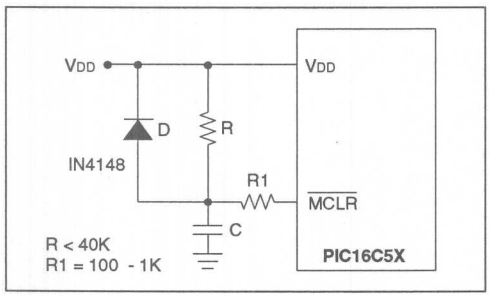
When VDD rises slowly, the internal time-out period expires long before VDD has reached its final value. In this example, the chip will reset properly if, and only if, V1≥ VDDMIN.

© 1993 Microchip Technology Inc.

EXTERNAL POWER-ON RESET CIRCUIT

To use power supplies with slow rise times it is necessary to use an external power-on reset circuit such as the one shown in Figure 5. This circuit uses an external RC to generate the reset pulse. The time constant of the RC should be long enough to guarantee that the reset pulse is still present until VDD has reached VDD min. R should be 40K or less to guarantee that the MCLR will pull to within 0.2 volts of VDD. (since the leakage spec on MCLR is ±5 µA, a resistor larger than 40K may cause input high voltage on this pin to be less than VDD – 0.2V, the required spec). The diode D is used to rapidly discharge the capacitor on power-down. This is very important as a power-up reset pulse is needed after a short power-down (less than the time constant of RC) or after a power spike. The resistor R1 protects against high current flowing into MCLR pin from fully charged capacitor C in the event MCLR pin breakdown is induced through ESD or EOS. The circuit, however, does not protect against brown-out situations where the power does not drop to zero, but merely dips below VDD MIN. In such a situation, voltage at the MCLR pin will not go low enough (i.e., below VIL) to guarantee a reset pulse. The following section presents an example circuit to protect against such brown-outs.

FIGURE 5 - EXTERNAL POWER-ON RESET CIRCUIT



BROWNOUT PROTECTION

In many applications it is necessary to guarantee a reset pulse whenever VDD is less than VDD min. This can be accomplished using a brownout protection circuit such as the one shown in Figure 6. This is a simple circuit that causes a reset pulse whenever VDD drops below the zener diode voltage plus the Vbe of Q1. A 3.3 volt zener will produce a reset pulse whenever VDD drops below about 4 volts. This circuit has a typical accuracy of about ±100 mV. A less expensive, albeit less precise, brownout circuit is shown in Figure 7. Transistor Q1 turns off when Vbe = VDD • R1/(R1+R2) falls below 0.7 V allowing R3 to pull down MCLR input.

FIGURE 6 - BROWNOUT PROTECTION CIRCUIT

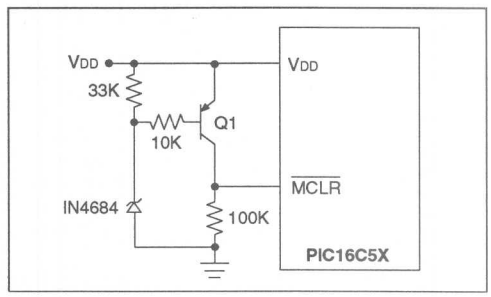
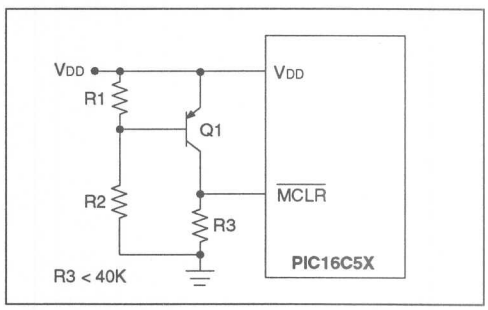


FIGURE 7 - BROWNOUT PROTECTION CIRCUIT



Author: Sumit Mitra
Logic Products Division

© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

Software Interrupt Techniques

INTRODUCTION

This application note describes a unique method for implementing interrupts in software on the PIC16C5X series of microcontrollers. The method takes advantage of the PIC16C5X's architecture which allows changing the program counter under software control. Up to eight interrupt lines are possible, but the practical limit for simple code generation is six interrupts, or 64 possible input conditions. The interrupt detection time is under software control and standard I/O pins are used as the interrupt lines.

THEORY OF OPERATION

SOFTWARE POLLING OF I/O LINES REPLACES HARDWARE INTERRUPT

The interrupt conditions are determined by detecting changes on the I/O lines that have been selected to be the interrupt lines. These changes are used to create a jump table that allows a different program response to each interrupt condition. The interrupt response time is under software control and can be as short as ten to twenty microseconds, depending on main program and interrupt subroutine program length.

CREATING THE INTERRUPT SUBROUTINE JUMP TABLE

Each I/O condition may have its own unique subroutine to respond to changes on the interrupt lines. Direct access to these routines is achieved by using the PIC16C5X's ability to change the program counter under software control. Here is an example of how two I/O lines may be polled:

```
MOVF   CONDTN,W  ;LOAD I/0 CONDITION INTO W
                 ;REGISTER
ANDLW  3         ;MASK OFF TOP 6 BITS
ADDWF  2,1       ;ADD INPUT TO PROGRAM COUNTER
                 ;TO CREATE JUMP TABLE
GOTO   MAIN      ;FOR  NO CHANGE GO TO MAIN
                 ;PROGRAM
GOTO   INT1      ;FOR CHANGE IN BIT 0 GOTO INT1
GOTO   INT2      ;FOR CHANGE IN BIT 1 GOTO INT2
GOTO   INT3      ;FOR BOTH CHANGE GOTO INT3
```

The changes to the I/O lines have been used to create a two bit number that is added to the program counter. The GOTO that is executed depends on the new program counter address.

CREATING CONSTANT TIME POLLING

In most applications requiring interrupts, it is important to poll the interrupt lines at fixed time intervals, usually only a few microseconds in length. Two techniques may be used on the PIC16C5X to achieve this. They are dividing the main program into multiple sections and implementing an elapsed time counter (see flow chart). Both of these techniques use the same program jump table concept that was described above. First, the main program is divided into several sections based on the desired I/O polling time. When MAIN is called a branch register is added to the program counter. This determines which section of MAIN code should be executed next. At the end of execution the branch register is decremented so the next section of code will be executed after the next polling. If the branch register is zero then the number of sections of main code is added into it to start the main program over again.

An elapsed time counter can be implemented using the RTCC counter. At the beginning of I/O polling the RTCC register is cleared. It then starts counting the instruction cycles. Then after the main program subsection has been executed, the RTCC register is subtracted from the desired polling time. This determines how many instructions need to be executed before the next polling. A jump table is then created to execute these instructions before the next polling. An example is shown below. This example assumes from zero to 15 additional instruction cycles are needed. Actual numbers need to be computed for each individual application.

```
MOVLW  POLL    ;POLL=DESIRED POLL CYCLES - 15
SUBWF  RTCC,W  ;DETERMINE HOW MUCH TIME TO WAIT
ADDWF  2,1     ;ADD  WAIT   TIME  TO  PROGRAM
COUNTER
NOP            ;15 ADDITIONAL INSTRUCTION CYCLES
:
:              ;TOTAL OF 15 NOP'S
NOP            ;1 ADDITIONAL INSTRUCTION CYCLES
GOTO   START   ;0 ADDITIONAL INSTRUCTION CYCLES
```

© 1993 Microchip Technology Inc.

For example, if the desired instruction time is 50 cycles and the subsection we just executed has a consumed a total of 40 instruction cycles (including all overhead cycles) the value of

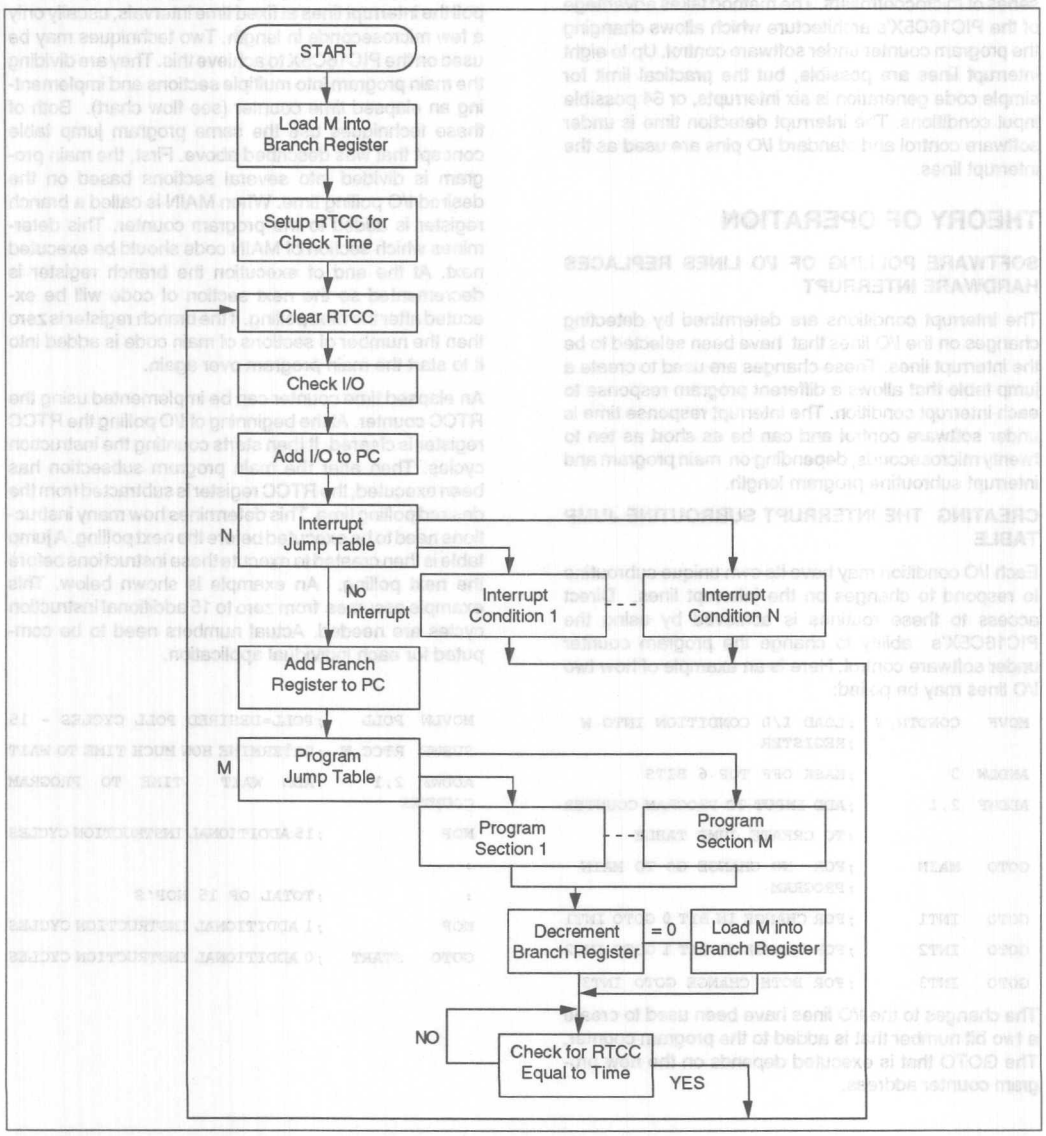
RTCC(40) - POLL(50-15(35)) =5

will be added to the program counter. The program will then jump to the sixth NOP. That NOP plus the 9 following it will be executed for a total of ten more instruction cycles. Note that the final GOTO has two instruction cycles and these must be included in the program overhead.

Example

The following example (see flow chart and code) is the core program for the software interrupt technique described above. This program assumes four interrupt conditions, four main program sections and an eight additional elapsed time instructions.

FIGURE 1 - SOFTWARE INTERRUPT FLOW CHART



© 1993 Microchip Technology Inc.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1



                      LIST    P=16C54

                                      ;SOFTWARE INTERRUPT APPLICATIONS
                                      ;BRANCH IS MAIN PROGRAM REGISTER
0008                  BRANCH  EQU     8
0009                  CNDTN   EQU     9
000A                  IO      EQU     0A
000B                  TEMP    EQU     0B

0000 0069             SETUP   CLRF    CNDTN
0001 0C04                     MOVLW   4
0002 0028                     MOVWF   BRANCH  ;FOUR MAIN PROGRAM SECTIONS
0003 0C08                     MOVLW   8
0004 0002                     OPTION          ;SET RTCC TO ONE COUNT PER INSTRUCTION CYCLE

0005 0061             START   CLRF    1       ;CLEAR RTCC REGISTER
0006 0206                     MOVF    6,W     ;READ I/O
0007 002A                     MOVWF   IO
0008 0109                     IORWF   CNDTN,W ;THIS SECTION OF CODE CALCULATES THE
0009 002B                     MOVWF   TEMP    ;JUMP TABLE. ANY INPUT THAT CHANGES FROM
000A 0209                     MOVF    CNDTN,W ;A ZERO TO A ONE IS CONSIDERED AN INTERRUPT.
000B 00AB                     SUBWF   TEMP,1  ;THE EQUATION IS:
000C 020A                     MOVF    IO,W    ;  (IO + CNDTN) - CNDTN = INTERRUPT
000D 0029                     MOVWF   CNDTN   ;WHERE IO IS CURRENT INPUT AND
000E 020B                     MOVF    TEMP,W  ;CNDTN IS PREVIOUS INPUT.
000F 0E03                     ANDLW   3       ;MASK OFF TOP 6 BITS
0010 01E2                     ADDWF   2,1     ;ADD INPUT TO PC TO CREATE JUMP TABLE
0011 0A1B                     GOTO    MAIN    ;FOR INPUT=00
0012 0A15                     GOTO    INT1    ;FOR INPUT=01
0013 0A17                     GOTO    INT2    ;FOR INPUT=10
0014 0A19                     GOTO    INT3    ;FOR INPUT=11

0015 0000             INT1    NOP             ;INTERRUPT LINE 1 CODE
0016 0A05                     GOTO    START
0017 0000             INT2    NOP             ;INTERRUPT LINE 2 CODE
0018 0A05                     GOTO    START
0019 0000             INT3    NOP             ;INTERRUPT LINES 1 AND 2 CODE
001A 0A05                     GOTO    START

001B 0208             MAIN    MOVF    BRANCH,W
001C 01E2                     ADDWF   2,1     ;ADD BRANCH TO PC TO CREATE JUMP TABLE
001D 0000                     NOP
001E 0A28                     GOTO    MAIN4   ;JUMP TABLE, LAST FIRST ON DECREMENT TABLE
001F 0A26                     GOTO    MAIN3
0020 0A24                     GOTO    MAIN2
0021 0A22                     GOTO    MAIN1

0022 0000             MAIN1   NOP             ;MAIN PROGRAM CODE BANK ONE
0023 0A2A                     GOTO    BRNCHK
0024 0000             MAIN2   NOP             ;MAIN PROGRAM CODE SECTION TWO
0025 0A2A                     GOTO    BRNCHK
0026 0000             MAIN3   NOP             ;MAIN PROGRAM CODE SECTION THREE
0027 0A2A                     GOTO    BRNCHK
0028 0000             MAIN4   NOP             ;MAIN PROGRAM CODE SECTION FOUR
0029 0A2A                     GOTO    BRNCHK

002A 02E8             BRNCHK  DECFSZ  BRANCH,1 ;DECREMENT BRANCH REGISTER AND CHECK FOR ZERO
002B 0A2E                     GOTO    TIMCHK
002C 0C04                     MOVLW   4
002D 0028                     MOVWF   BRANCH  ;RELOAD BRANCH WITH 4 AT END OF MAIN

002E 0C29             TIMCHK  MOVLW   D'41'   ;CHECK TO SEE IF RTCC HAS REACHED 50(50-7)
```

© 1993 Microchip Technology Inc.

```
002F 0081                SUBWF   1,W     ;DETERMINE WAIT TIME
0030 01E2                ADDWF   2,1     ;ADD WAIT TIME TO PC
0031 0000                NOP
0032 0000                NOP
0033 0000                NOP
0034 0000                NOP
0035 0000                NOP
0036 0000                NOP
0037 0000                NOP
0038 0A05                GOTO    START
                         END


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

Software Stack Management

INTRODUCTION

The PIC16C5X has a stack which is only 2 deep, as a result of which only two nested calls can be made (i.e. only one call within a call routine). If more than two levels of subroutine nesting is required, this application note can be used to implement a stack manager to handle the flow of the calls.

Note: Since the amount of RAM on the PIC16CXX is limited, it would be prudent to determine the maximum number of nested calls which have to be made in a program and define the stack length appropriately.

IMPLEMENTATION

This application note implements a 5-deep stack, so 5 nested calls can be made without overflowing the stack. NCALL is defined as a MACRO which will be used instead of the mnemonic CALL, when a subroutine call is made. The NCALL routine, "pushes" the return PC value on the "stack" and then executes the called subroutine. At the end of the subroutine, instead of using the RETLW k instruction, a GOTO RETURN is executed, where RETURN is a routine which "pops" the return PC value from the "stack" and resumes the normal flow of the program.

Note: Since Software Stack Management utilizes the FSR register, and indirect addressing, the user should restore the "original" values to the FSR register if it is utilized elsewhere in the program.

The routines, as described in this application note, will work only if the called routine is within the first 256 words for each program. If the user desires to branch over to the other low 256-byte program pages, as in the PIC16C57, then the status byte should be saved along with the PC.

Author: Stanley D'Souza
Logic Products Division

© 1994 Microchip Technology Incorporated

```
MPASM 1.00 Released       SM.ASM   7-15-1994  13:9:35                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         list p=16c54,f=inhx8m
                     0002 ;*************************************************
                     0003 ;       sm.asm:
                     0004 ;       Routine, demonstrating how to implement a stack
                     0005 ;       manager capable of handling more than 2
                     0006 ;       subsequent subroutine calls.
                     0007 ;       Note: Since this is a demo, NOP has been used
                     0008 ;       where normally the body of the subroutine would
                     0009 ;       reside.
                     0010 ;*************************************************
                     0011 ;
0002                 0012         PC      EQU     2
0004                 0013 FSR     EQU     4
                     0014 ;
                     0015 ;
0008                 0016 STACK   EQU     8       ;define stack top
                     0017 ;*****************************************************************
                     0018 ;NOTE: the next 5 locations in RAM should be reserved for the
                     0019 ;"STACK" implementation. Please do not use any ram locations
                     0020 ;from decimal 8 to decimal 12.
                     0021 ;*****************************************************************
                     0022 ;
                     0023         ORG     01FF
01FF 0A07            0024         GOTO    START
                     0025 ;
                     0026         ORG     0
                     0027 ;
0000 0C08            0028 INIT    MOVLW   STACK   ;load "stack" as indirect pointer
0001 0024            0029         MOVWF   FSR     ;       /
0002 0A07            0030         GOTO    START   ;      /
                     0031 ;
                     0032 ;*************************************************
                     0033 ;define NCALL as a MACRO used instead of the
                     0034 ;mnemonic CALL.
                     0035 ;
                     0036 NCALL   MACRO   LABEL
                     0037         MOVF    PC,W    ;save PC on "stack"
                     0038         MOVWF   0       ;       /
                     0039         INCF    FSR     ;Inc. "stack"  pointer.
                     0040         GOTO    LABEL   ;jump to routine
                     0041         ENDM
                     0042 ;
                     0043 ;return from subroutine NCALL
                     0044 ;
0003 00E4            0045 RETURN  DECF    FSR     ;point to last "stack" location
0004 0C03            0046         MOVLW   3       ;add 3 and output value from FSR
0005 01C0            0047         ADDWF   0,W     ;       /
0006 0022            0048         MOVWF   PC      ;load in PC as next executable
                     0049 ;                       instruction
                     0050 ;
                     0051 ;*************************************************
                     0052 ;
```


© 1994 Microchip Technology Incorporated

```
MPASM 1.00 Released       SM.ASM   7-15-1994  13:9:35          PAGE  2


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0054 ;
0007 0000            0055 START   NOP
                     0056         NCALL   TOM
0008 0202               M         MOVF    PC,W    ;save PC on "stack"
0009 0020               M         MOVWF   0       ;       /
000A 02A4               M         INCF    FSR     ;Inc. "stack" pointer.
000B 0A0F               M         GOTO    TOM   ;jump to routine
000C 0000            0057         NOP     ;body of main routine
000D 0000            0058         NOP     ;       /
000E 0003            0059         SLEEP
                     0060 ;
000F 0000            0061 TOM     NOP
                     0062         NCALL   DICK
0010 0202               M         MOVF    PC,W    ;save PC on "stack"
0011 0020               M         MOVWF   0       ;       /
0012 02A4               M         INCF    FSR     ;Inc. "stack" pointer.
0013 0A16               M         GOTO    DICK   ;jump to routine
0014 0000            0063         NOP     ;body of routine TOM
0015 0A03            0064         GOTO    RETURN
                     0065 ;
0016 0000            0066 DICK    NOP
                     0067         NCALL   HARRY
0017 0202               M         MOVF    PC,W    ;save PC on "stack"
0018 0020               M         MOVWF   0       ;       /
0019 02A4               M         INCF    FSR     ;Inc. "stack" pointer.
001A 0A1D               M         GOTO    HARRY   ;jump to routine
001B 0000            0068         NOP     ;body of routine DICK
001C 0A03            0069         GOTO    RETURN
                     0070 ;
001D 0000            0071 HARRY   NOP     ;body of routine HARRY
001E 0000            0072         NOP     ;       /
001F 0A03            0073         GOTO    RETURN
                     0074 ;
                     0075 ;
                     0076         END
                     0077
                     0078
```


© 1994 Microchip Technology Incorporated

```
MPASM 1.00 Released  SM.ASM   7-15-1994  13:9:35          PAGE  3


SYMBOL TABLE

LABEL                             VALUE

DICK                              0016
FSR                               0004
HARRY                             001D
INIT                              0000
PC                                0002
RETURN                            0003
STACK                             0008
START                             0007
TOM                               000F


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------

0180 : ---------------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Incorporated

AN581

Implementing Long Calls

INTRODUCTION

This application note discusses how to implement "long Calls" in the PIC16C5X architecture. The use of long call can simplify the partitioning of the application program with minimal software overhead.

In the PIC16C5X architecture, the program memory page size is 512 words. Depending on the device, the program memory may be as large as 2K words (as in the PIC16C57 or PIC16C58 devices). The program counter (and stack) width range from 9- to 11-bits, depending on the amount of program memory the device has. Table 1 shows the width of the Program Counter (PC) and Stack for the various devices.

TABLE 1: PC AND STACK WIDTH

Device	Width (Bits)		Program Memory (Words)
	Program Counter	Stack	
PIC16C54 / PIC16C55	9	9	512
PIC16C56	10	10	1K
PIC16C57 / PIC16C58	11	11	2K

The low order 8-bits of the program counter are accessible by the user program. These bits are contained in the PC register. The entire Program Counter is shown in Figure 1.

Since A8 is forced to 0 by CALL instructions, the start address of subroutines must be in the first 256 words of each program memory page. Depending on the size and number of called subroutines, this limitation may become a burden to the software developer. The implementation of a "long call" eases this, by allowing the subroutine to be anywhere in the program memory page. The three important concepts, to understand the implementation of the long call are:

1. A CALL instruction loads the entire PC onto the Stack
2. A GOTO instruction does not affect the Stack
3. A GOTO instruction can branch to any location in a program memory page.

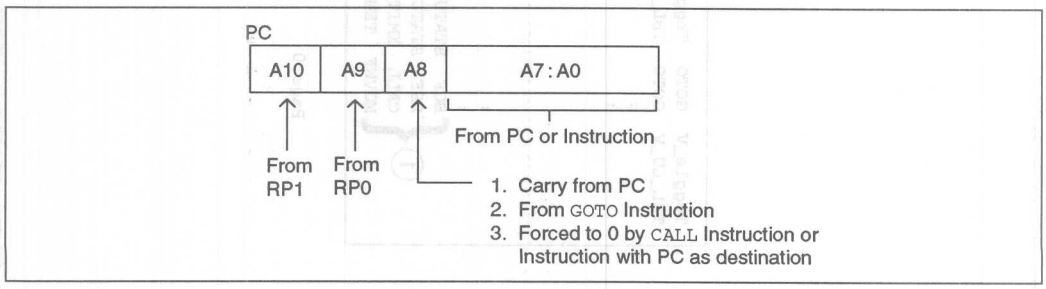
Also to select the desired page, the RP1 and RP0 bits (STATUS<6:5>) must be programmed accordingly. These bits do not get loaded into A10:A9 , of the PC, until one of the following occurs:

1. A CALL instruction
2. A GOTO instruction
3. An instruction that modifies the PC register (PC<:7:0>), such as ADDWF PC, F.

So a CALL instruction followed by a GOTO instruction will always remain in the same page as the intended call. This allows the developer to place "call vectors" at the first 256 words of each page. The instruction at the "call vector" then executes a GOTO instruction to the subroutine anywhere in that page. The RETLW instruction, of the subroutine, will then POP the stack. The Stack contained the PUSHed PC from the CALL instruction.

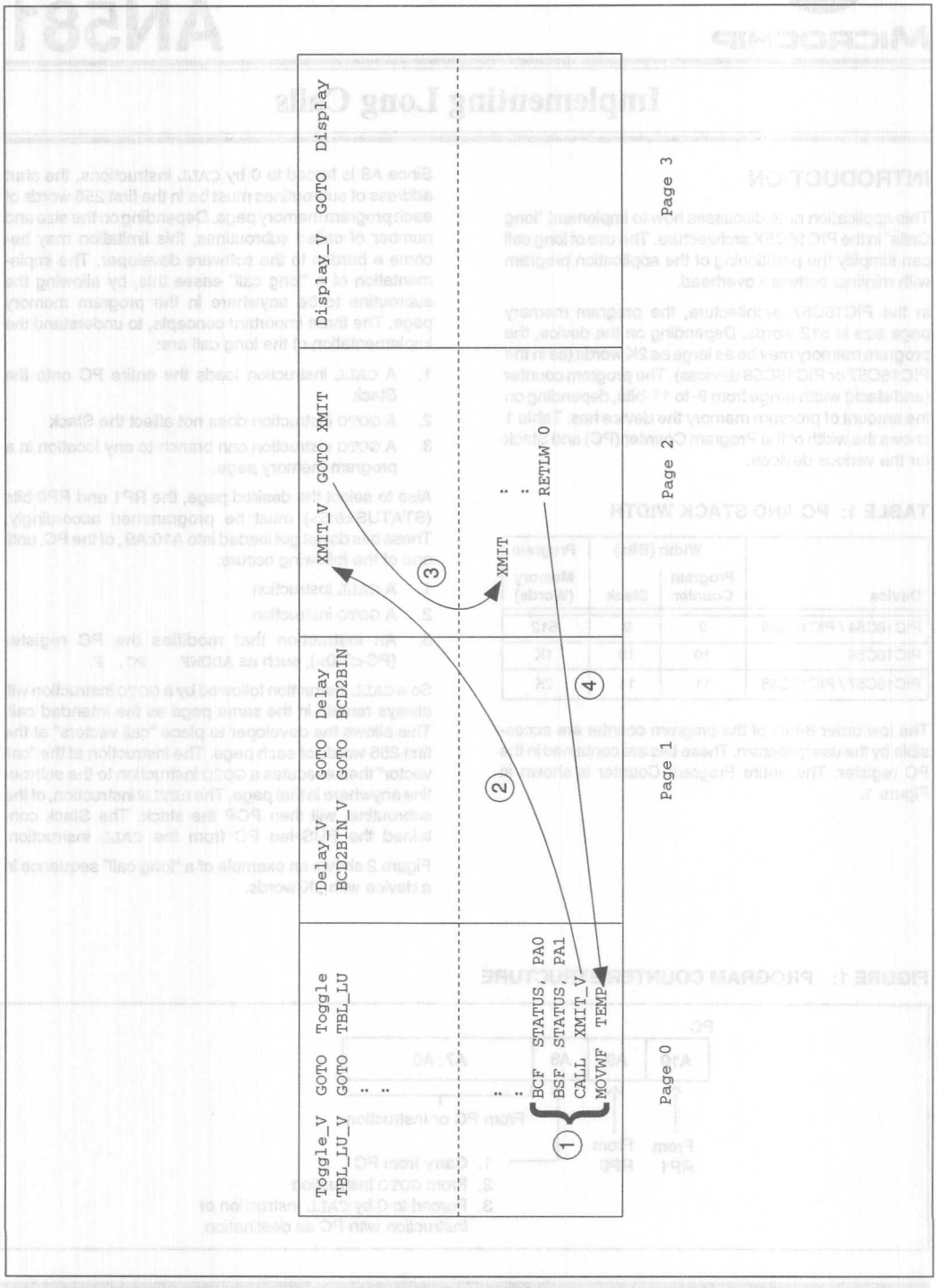
Figure 2 shows an example of a "long call" sequence in a device with 2K-words.

FIGURE 1: PROGRAM COUNTER STRUCTURE



© 1994 Microchip Technology Inc.

FIGURE 2: EXAMPLE OF A "LONG CALL" SEQUENCE



© 1994 Microchip Technology Inc.

The flow that occurs in Figure 2 is as follows:

1. Select the program memory page of the desired subroutine and execute the call to that subroutine.
2. The program loads the Stack with the PC+1 address, branches to the selected page and specified address of the "call vector" (must be in the first 256 locations of the page)
3. Executes a GOTO instruction, to have access to the entire program memory page. Then executes the subroutine.
4. Executes the RETLW instruction, which POPs the new PC from the Stack. This causes program execution to continue at the instruction after the CALL instruction.

The use of "long calls" could be used to place all the subroutines in selected page(s), since the entire page can contain the subroutines (not restricted to the top half of the page). The placing of all subroutines in fewer program memory pages can reduce the overhead of specifying the required pages, since they are changed less frequently.

Use of the MPASM assembler can ease in the verification that call vectors and the call routine are in the same program memory page. Example 1 shows the use of assembler directives to print user defined warning or error messages in the listing file. These are shown as the shaded conditional statements. These messages are only printed in the listing file, and no indication of these messages is shown at the completion of assembly.

© 1994 Microchip Technology Inc.

EXAMPLE 1: USE OF ASSEMBLER DIRECTIVES

```
;
P1_TOP                  EQU             0x0000          ; First address in page 0
P2_TOP                  EQU             0x0200          ; First address in page 1
P3_TOP                  EQU             0x0400          ; First address in page 2
P4_TOP                  EQU             0x0600          ; Reset vector address in page 1
RESET_V                 EQU             0x07FF
;
    org     P1_TOP
;
     :
     :
     :
;
    org     P3_TOP
;
My_Subroutine_V     GOTO    My_Subroutine    ; Vector for My_Subroutine
     :
     :
     :
;
My_Subroutine                                ; My_Subroutine routine
;
   if ( ( My_Subroutine_V & 0x0600 ) != ( My_Subroutine & 0x0600 ) )
      MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
   endif
;
     :
     :
     :
;
My_Subroutine_END    RETURN
;
   if ( ( My_Subroutine_V & 0x0600) != (My_Subroutine_END & 0x0600) )
      MESSG   "Warning - User Defined: Call routine crosses page boundry"
   endif
;
     :
     :
     :
;
       org   RESET_V                         ; Program memory address for the reset vector
;
             GOTO          START             ; Goto the beginning of the program
```

CONCLUSION

The use of "long calls" may ease the development of application programs. For minimal overhead, the application program can execute a subroutine from anywhere in the program memory, and return to the desired location. This eases the development of the application program, by reducing the mapping of subroutine in the first 256 words of each program memory page. The use of "long calls" is possible in any of the PIC16C5X devices, but is most useful in the devices with more than one program memory page. For device with more than one page of program memory, the assembler directives can be used to verify that the subroutines are in the program memory page.

Author: Mark Palmer - Sr. Application Engineer

© 1994 Microchip Technology Inc.

AN586

Macros for Page and Bank Switching

INTRODUCTION

This application note discusses the use of the MPASM assembler's conditional assembly to automatically switch between program memory pages or to set the data memory banks. These macros, along with the long call technique (see Application Note AN581), ease the development of software. Though the use of these macros can simplify the program memory paging and data memory banking with minimal software overhead. The use of these macros without thought can causes unnecessary (duplicate) instructions to be used, by setting page or bank bits unnecessarily.

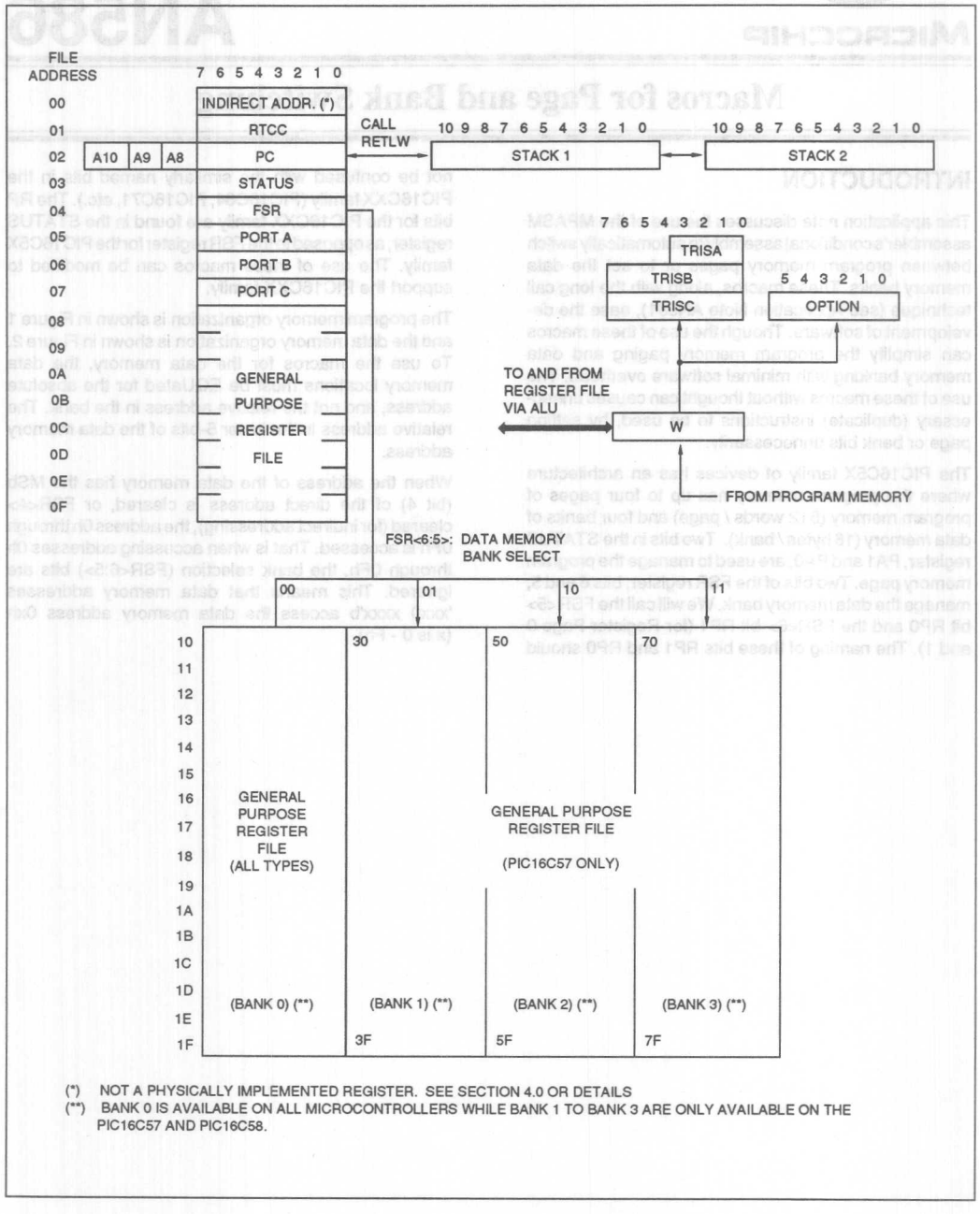
The PIC16C5X family of devices has an architecture where the program memory has up to four pages of program memory (512 words / page) and four banks of data memory (16 bytes / bank). Two bits in the STATUS register, PA1 and PA0, are used to manage the program memory page. Two bits of the FSR register, bits 6 and 5, manage the data memory bank. We will call the FSR<5> bit RP0 and the FSR<6> bit RP1 (for Register Page 0 and 1). The naming of these bits RP1 and RP0 should not be confused with the similarly named bits in the PIC16CXX family (PIC16C64, PIC16C71, etc.). The RP bits for the PIC16CXX family are found in the STATUS register, as opposed to the FSR register for the PIC16C5X family. The use of these macros can be modified to support the PIC16CXX family.

The program memory organization is shown in Figure 1 and the data memory organization is shown in Figure 2. To use the macros for the data memory, the data memory locations must be EQUated for the absolute address, and not the relative address in the bank. The relative address is the lower 5-bits of the data memory address.

When the address of the data memory has the MSb (bit 4) of the direct address is cleared, or FSR<4> cleared (for indirect addressing), the address 0h through 0Fh is accessed. That is when accessing addresses 0h through 0Fh, the bank selection (FSR<6:5>) bits are ignored. This means that data memory addresses 'xxx0 xxxx'b access the data memory address 0xh (x is 0 - Fh).

© 1994 Microchip Technology Inc.

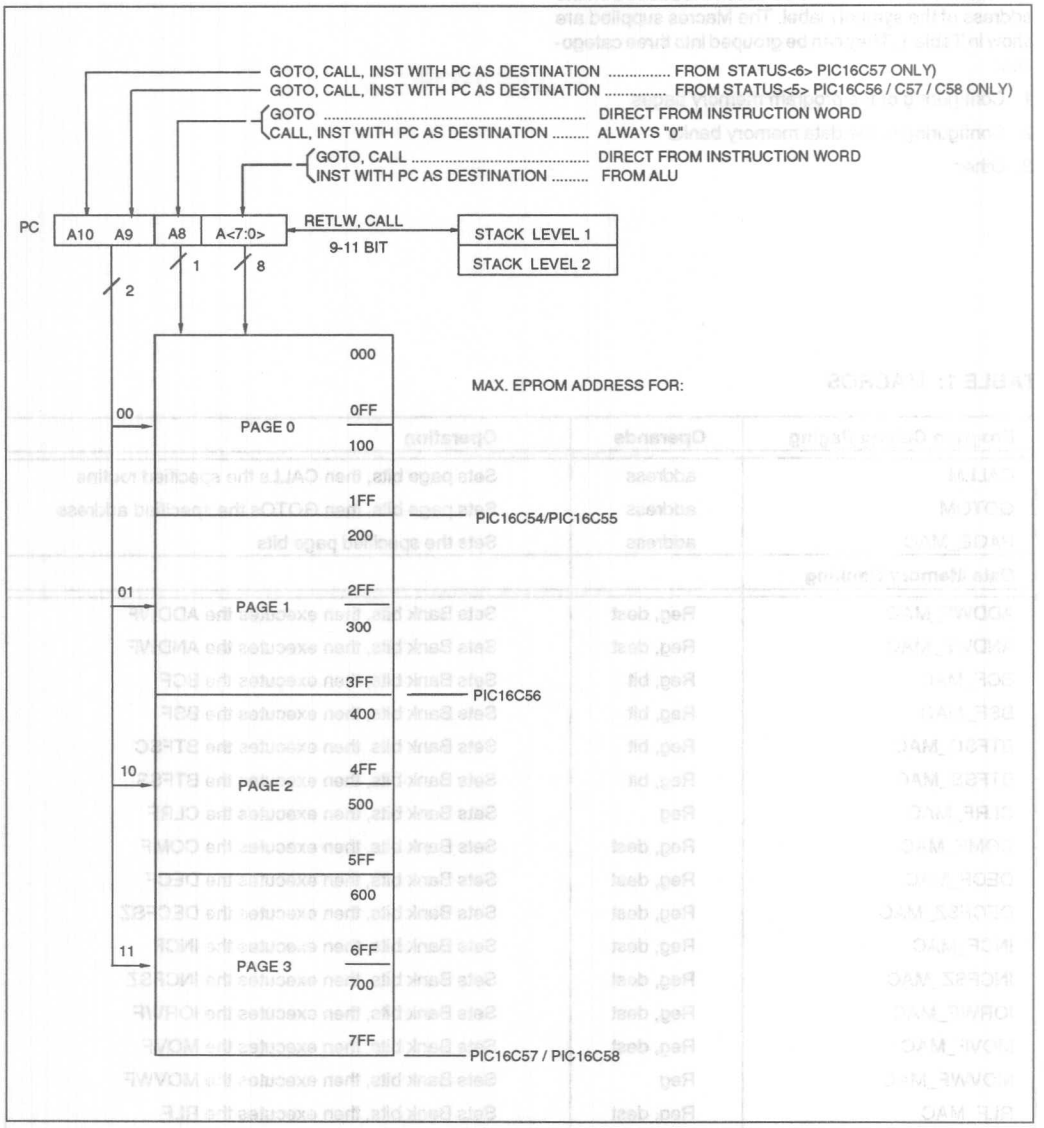
FIGURE 1: PROGRAM MEMORY ORGANIZATION



(*) NOT A PHYSICALLY IMPLEMENTED REGISTER. SEE SECTION 4.0 OR DETAILS
(**) BANK 0 IS AVAILABLE ON ALL MICROCONTROLLERS WHILE BANK 1 TO BANK 3 ARE ONLY AVAILABLE ON THE PIC16C57 AND PIC16C58.

© 1994 Microchip Technology Inc.

FIGURE 2: DATA MEMORY MAP



© 1994 Microchip Technology Inc.

Macros for Page and Bank Switching

The use of MPASM's conditional assembly, allows the selection of source code to be assembled based on the address of the symbol / label. The Macros supplied are show in Table 1. They can be grouped into three categories:

1. Configuring of the program memory pages
2. Configuring of the data memory banks
3. Other

TABLE 1: MACROS

Program Calling Paging	Operands	Operation
CALLM	address	Sets page bits, then CALLs the specified routine
GOTOM	address	Sets page bits, then GOTOs the specified address
PAGE_MAC	address	Sets the specified page bits
Data Memory Banking		
ADDWF_MAC	Reg, dest	Sets Bank bits, then executes the ADDWF
ANDWF_MAC	Reg, dest	Sets Bank bits, then executes the ANDWF
BCF_MAC	Reg, bit	Sets Bank bits, then executes the BCF
BSF_MAC	Reg, bit	Sets Bank bits, then executes the BSF
BTFSC_MAC	Reg, bit	Sets Bank bits, then executes the BTFSC
BTFSS_MAC	Reg, bit	Sets Bank bits, then executes the BTFSS
CLRF_MAC	Reg	Sets Bank bits, then executes the CLRF
COMF_MAC	Reg, dest	Sets Bank bits, then executes the COMF
DECF_MAC	Reg, dest	Sets Bank bits, then executes the DECF
DECFSZ_MAC	Reg, dest	Sets Bank bits, then executes the DECFSZ
INCF_MAC	Reg, dest	Sets Bank bits, then executes the INCF
INCFSZ_MAC	Reg, dest	Sets Bank bits, then executes the INCFSZ
IORWF_MAC	Reg, dest	Sets Bank bits, then executes the IORWF
MOVF_MAC	Reg, dest	Sets Bank bits, then executes the MOVF
MOVWF_MAC	Reg	Sets Bank bits, then executes the MOVWF
RLF_MAC	Reg, dest	Sets Bank bits, then executes the RLF
RRF_MAC	Reg, dest	Sets Bank bits, then executes the RRF
SUBWF_MAC	Reg, dest	Sets Bank bits, then executes the SUBWF
SWAPF_MAC	Reg, dest	Sets Bank bits, then executes the SWAPF
XORWF_MAC	Reg, dest	Sets Bank bits, then executes the XORWF
BANK_MAC	Reg	Sets the specified Bank bits
Other		
SAVE_W_STATUS	-	Saves the W and STATUS registers
RESTORE_W_STATUS	-	Restores the W and STATUS registers

© 1994 Microchip Technology Inc.

These macros (see Appendix A) ease the development of programs, but care should be taken in their use so that redundant instructions are not caused. An example of this is if you wanted to do the operations, INCF and BTFSS, on data memory location CNTR (in bank 3) and the FSR was pointing to some other bank. The use of the macros for both operations would cause six program memory locations to be assembled, while with some thought only four words are needed (see Example 1).

CONCLUSION

The use of these macros simplify the program development by managing the memory resources of the PIC16C5X device. If the application program becomes too large for the desired device program memory, it is recommended to study the listing file for any unnecessary code due to non-optimum usage of these macros. The MAC_TST.ASM file, is supplied to show how these macros work in a program.

EXAMPLE 1A: GENERATION OF UNNECESSARY CODE

```
INCF_MAC        CNTR, F         ->      BSF     FSR, 5
                                        BSF     FSR, 6
                                        INCF    CNTR, F
BTFSS_MAC       CNTR, 5         ->      BSF     FSR, 5          ; Unnecessary, already in bank
                                        BSF     FSR, 6          ; Unnecessary, already in bank
                                        BTFSS   CNTR, 5
```

EXAMPLE 1B: GENERATION OF OPTIMUM CODE

```
INCF_MAC        CNTR, F         ->      BSF     FSR, 5
                                        BSF     FSR, 6
                                        INCF    CNTR, F
BTFSS           CNTR, 5         ->      BTFSS   CNTR, 5
```

Written By: Mark Palmer - Sr. Application Engineer
Contributions by: Mike Morse - Sr. Field Application Engineer (Dallas)

© 1994 Microchip Technology Inc.

APPENDIX A: MACRO FILE

```
  nolist
;*************************************************************************
; This file contains MACROs to ease in the use of the Program Memory
; paging and the Data Memory bank switching for the PIC16C5x devices
;
;    File Name:    AUTO_PG.MAC
;    REVISION:     5-20-94
;*************************************************************************
;
PAGE1_OR_3    EQU       0x0200          ; Program Memory in page 1 or page 3
PAGE2_OR_3    EQU       0x0400          ; Program Memory in page 2 or page 3
;
BANK1_OR_3    EQU       0x020           ; Data Memory in Bank 1 or Bank 3
BANK2_OR_3    EQU       0x040           ; Data Memory in Bank 2 or Bank 3
;
;*************************************************************************
;**                          CALLM     program_address
;** Configures the PA1 and PA0 bits as required, ensures that the CALLed
;** "routine" is in the first 256 locations of the program memory page.
;** If the "routine" is in the second 256 locations of the program memory page,
;** an User Defined ERROR Message is placed in the LISTING file. MPASM
;** presently only places this message in the listing file (i.e. no indication
;** is shown when MPASM completes execution in the ERROR / WARNING listed.
;**
;*************************************************************************
;
CALLM          macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
     if ( ( routine & 0x0100 ) == 0x0100 )
       MESSG "Error - User Defined: CALLed routine in 2nd 256 locations of the
            program memory page"
     endif
;
               CALL    routine
               endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**                          GOTOM    program_address
;** Configures the PA1 and PA0 bits as required, and GOTOs the specified
;** locations of the program memory page.
;**
;*****************************************************************************
;
GOTOM       macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
               GOTO    routine
               endm
;
;*****************************************************************************
;**                    ADDWF_MAC      data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "ADDWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
ADDWF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               ADDWF    address, d
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                  ANDWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "ANDWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
ANDWF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
               ANDWF    address, d
               endm
;
;*******************************************************************************
;**                  BCF_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "BCF data_address, bit" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
BCF_MAC         macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
               BCF     address, b
               endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**                  BSF_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "BSF data_address, bit" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
BSF_MAC         macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BSF     address, b
               endm
;
;*****************************************************************************
;**                  BTFSC_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "BTFSC data_address, bit" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
BTFSC_MAC       macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BTFSC     address, b
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                  BTFSS_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "BTFSS data_address, bit" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
BTFSS_MAC       macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
                BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
                BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
                BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
                BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
                BTFSS    address, b
                endm
;
;*******************************************************************************
;**                  CLRF_MAC     data_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "CLRF data_address" instruction. The data_address
;** must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
CLRF_MAC        macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
                BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
                BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
                BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
                BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
                CLRF    address
                endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**                    COMF_MAC    data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "COMF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
COMF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               COMF    address, d
               endm
;
;*****************************************************************************
;**                    DECF_MAC    data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "DECF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
DECF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               DECF    address, d
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                  DECFSZ_MAC      data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "DECFSZ data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
DECFSZ_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF       FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF       FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF       FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF       FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               DECFSZ    address, d
               endm
;
;*******************************************************************************
;**                  INCF_MAC        data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "INCF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
INCF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF       FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF       FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF       FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF       FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               INCF      address, d
               endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**                    INCFSZ_MAC      data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "INCFSZ data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
INCFSZ_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               INCFSZ  address, d
               endm
;
;*****************************************************************************
;**                    IORWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "IORWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
IORWF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               IORWF    address, d
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                    MOVF_MAC      data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "MOVF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
MOVF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
                BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
                BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
                BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
                BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
                MOVF    address, d
                endm
;
;*******************************************************************************
;**                    MOVWF_MAC     data_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "BSF data_address" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
MOVWF_MAC       macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
                BSF     FSR, RP0            ; Set RP0 for Data Memory Page
     else
                BCF     FSR, RP0            ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
                BSF     FSR, RP1            ; Set RP1 for Data Memory Page
     else
                BCF     FSR, RP1            ; Clear RP1 for Data Memory Page
     endif
                MOVWF    address
                endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**          RLF_MAC   data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "RLF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
RLF_MAC         macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               RLF     address, d
               endm
;
;*****************************************************************************
;**          RRF_MAC   data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "RRF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
RRF_MAC         macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               RRF     address, d
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                SUBWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "SUBWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
SUBWF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               SUBWF   address, d
               endm
;
;*******************************************************************************
;**                SWAPF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "SWAPF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
SWAPF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               SWAPF   address, d
               endm
```


© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                  XORWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "XORWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
XORWF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               XORWF   address, d
               endm
;
;*******************************************************************************
;**                  PAGE_MAC      program_address
;** Configures the PA1 and PA0 bits as required
;**
;*******************************************************************************
;
PAGE_MAC          macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
               endm
```

© 1994 Microchip Technology Inc.

```
;
;*******************************************************************************
;**                  BANK_MAC      program_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing.
;** The data_address must be the absolute address and NOT the relative address
;** in the data memory page.
;**
;*******************************************************************************
;
BANK_MAC        macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               endm
;
;*******************************************************************************
;**                  SAVE_W_AND_STATUS
;** Saves the contects of the W register and the STATUS register to two
;** temporary RAM locations W_TEMP and STATUS_TEMP. These temporary RAM
;** locations should be in the NON-Banked part of Data Memory (8h to Fh).
;** This Macro generates a User Defined Warning (seen only in listing file)
;** if the Data RAM location is in Banked RAM.
;**
;*******************************************************************************
;
SAVE_W_AND_STATUS       macro
;
               MOVWF   W_TEMP
               SWAPF   W_TEMP, F
               SWAPF   STATUS, W
               MOVWF   STATUS_TEMP
     if ( ( W_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: W_TEMP register is defined to be in BANKed
             memory"
     endif
;
     if ( ( STATUS_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: STATUS_TEMP register is defined to be in BANKed
             memory"
     endif
               endm
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************
;**                    RESTORE_W_AND_STATUS
;** Saves the contects of the W register and the STATUS register to two
;** temporary RAM locations W_TEMP and STATUS_TEMP. These temporary RAM
;** locations should be in the NON-Banked part of Data Memory (8h to Fh).
;** This Macro generates a User Defined Warning (seen only in listing file)
;** if the Data RAM location is in Banked RAM.
;**
;*****************************************************************************
;
RESTORE_W_AND_STATUS      macro
;
               SWAPF   STATUS_TEMP, W
               MOVWF   STATUS
               SWAPF   W_TEMP, W
     if ( ( W_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: W_TEMP register is defined to be in BANKed
             memory"
     endif
;
     if ( ( STATUS_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: STATUS_TEMP register is defined to be in BANKed
             memory"
     endif
               endm
;

     list
```

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

AN528

Implementing Wake-Up on Key Stroke

INTRODUCTION

In certain applications, the PIC16CXX is exercised only when a key is pressed, eg. remote keyless entry. In such applications, the battery life can be extended by putting the PIC16CXX to sleep during the inactive state and when a key is pressed, the PIC16CXX wakes up, does the task, and then goes back to sleep.

IMPLEMENTATION

The circuit in Figure 1 depicts an application with two keys. The PIC16C54 is normally in SLEEP mode consuming very little operating current. If either of the two keys is pressed, the PIC16C5X 'wakes up', scans the keys and turns on one of the two LED's. When SW1 is pressed, the green LED is turned on and when SW2 is pressed the red LED is turned on. The LED's are used purely for demonstration purposes. In real life application, a transmission will be completed before putting the PIC16C5X back in sleep. This example can be extended to handle more than two keys.

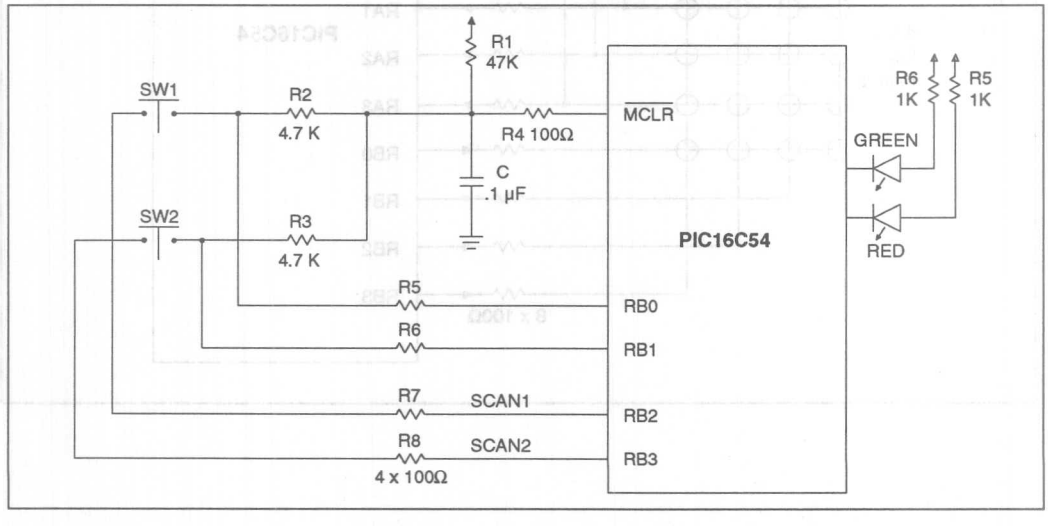
In the sleep mode, the scan outputs (SCAN1 and SCAN2) are both set to a low logic level. In this state, the capacitor C is fully charged and a high logic level is present at the $\overline{MCLR}$ pin of the PIC16C5X. When a key is pressed, C discharges through either R2 or R3 (depending on SW1 or SW2 being pressed) and the voltage across C falls rapidly (approx. 1 ms), causing a low at the $\overline{MCLR}$ pin of the PIC16C5X, which in turn causes the PIC16C5X to wake up and enter its reset state. In reset, the SCAN1 and SCAN2 outputs default to a high impedance mode, so the discharge path for capacitor C is blocked and it charges to a high level through resistor R1. Note that the RC values have been chosen such, that the discharge and charge cycles times are less than the reset time for the PIC16C5X (approx. 18 ms), and certainly far less than the minimum duration of a keypress (approx. 50-100 ms).

After the reset cycle is completed, the code execution momentarily takes the SCAN1 and SCAN2 outputs low in order to sample the key stroke(s). This does not cause the capacitor to discharge since the duration of the low is of the order of 10 micro seconds.

Once the keystroke function has been executed, the program loops until the key has been released, sets the SCAN1 and SCAN2 outputs low and "goes back to sleep". Resistors R4-R8 are not required for functionality, but are recommended to provide protection from electrostatic discharge (ESD). Switches SW1 and SW2, when pressed may frequently pass ESD to the PIC16C54.

Author: Stan D'Souza
Logic Products Division

FIGURE 1 - TWO KEY INTERFACE TO PIC16C5X



© 1993 Microchip Technology Incorporated

FIGURE 2 - TWO KEY SCAN/WAKE-UP TIMING DIAGRAM

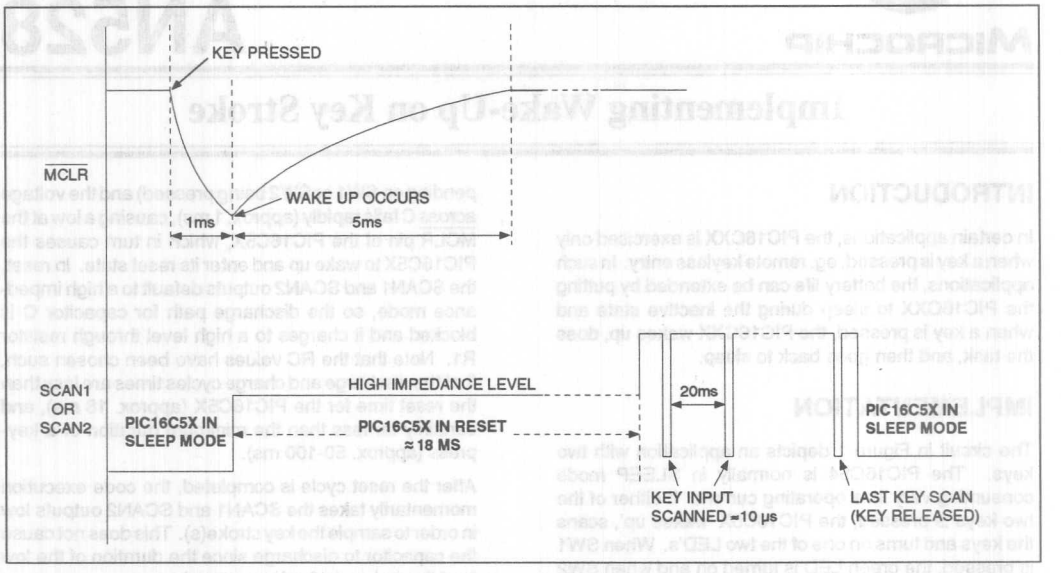
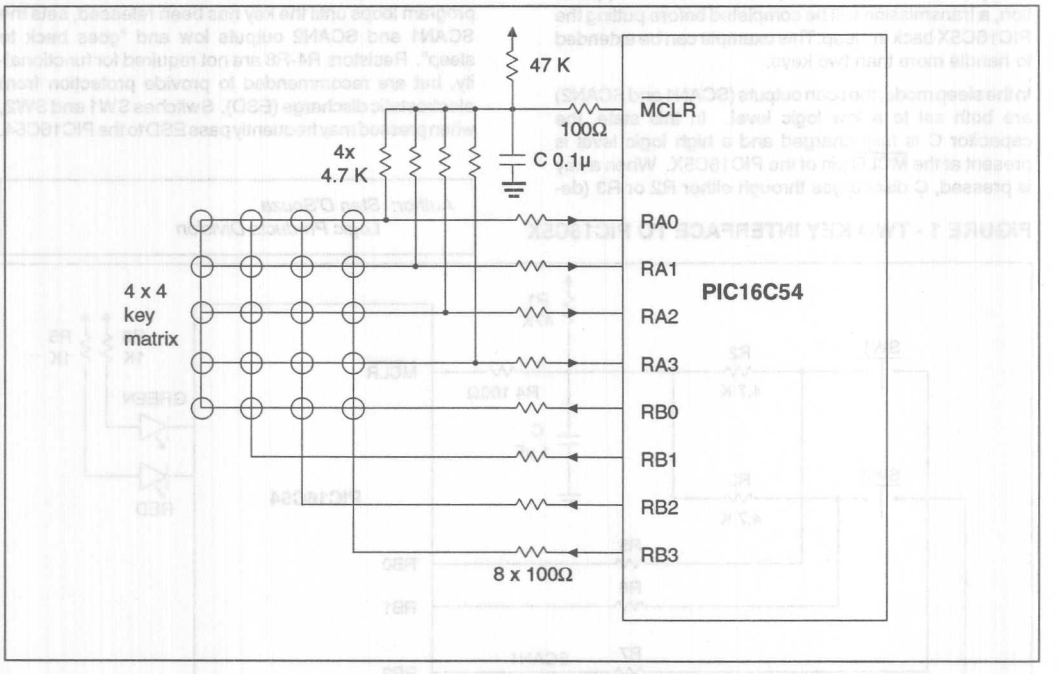


FIGURE 3 - PIC16C5X INTERFACE TO 4 X 4 KEY MATRIX



© 1993 Microchip Technology Incorporated

```
MPASM 1.00 Released        WU.ASM   7-15-1994  13:10:54                PAGE  1
Key Stroke Wake Up

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001            TITLE       "Key Stroke Wake Up"
                     0002          LIST  P = 16C54,f=inhx8m
                     0003 ;*************************************************
                     0004 ;  Program demonstrating key stroke wake up for
                     0005 ;  the PIC16CXX. Program has been implemented for
                     0006 ;  two keys, but can be extended for more keys.
                     0007 ;  When SW1 is pressed a green LED lights up.
                     0008 ;  When SW2 is pressed a red LED lights up.
                     0009 ;*************************************************
                     0010 ;
                     0011 ; Define equates
                     0012 ;
0002                 0013          PC        EQU    2
0006                 0014          PORT_B    EQU    6
0002                 0015          SCAN1     EQU    2
0003                 0016          SCAN2     EQU    3
0000                 0017          SW1       EQU    0
0001                 0018          SW2       EQU    1
0004                 0019          GRN_LED   EQU    4
0005                 0020          RED_LED   EQU    5
0014                 0021          MSEC_20   EQU    D'20'
0008                 0022          DB1       EQU    8
0008                 0023          GP        EQU    8
0009                 0024          DB2       EQU    9
                     0025 ;
                     0026 ;PORT_B ASSIGNMENTS:
                     0027 ;        0 -> SW1        INPUT
                     0028 ;        1 -> SW2        INPUT
                     0029 ;        2 -> SCAN1      OUTPUT
                     0030 ;        3 -> SCAN2      OUTPUT
                     0031 ;        4 -> GRN_LED    OUTPUT
                     0032 ;        5 -> RED_LED    OUTPUT
                     0033 ;      6&7 -> ASSIGNED AS DUMMY OUTPUTS
                     0035 ;
                     0036 ;
                     0037          ORG       0
                     0038 ;
                     0039 START
0000 0910            0040          CALL    INIT_PORT_B     ;INITIALIZE PORT B
0001 0920            0041          CALL    DELAY           ;DELAY 20 MSECS
0002 0915            0042          CALL    SCAN_KEYS       ;GET KEY VALUES
0003 0028            0043          MOVWF   GP              ;SAVE IN RAM
0004 0608            0044          BTFSC   GP,SW1          ;SKIP IF SW1 NOT PRESSED
0005 0929            0045          CALL    TURN_GREEN_ON   ;ELSE DO ROUTINE
0006 0628            0046          BTFSC   GP,SW2          ;SKIP IF SW2 NOT PRESSED
0007 092B            0047          CALL    TURN_RED_ON     ;ELSE DO ROUTINE
                     0048 CHK_FOR_KEY
0008 0920            0049          CALL    DELAY           ;DELAY FOR 20 MSEC
0009 0915            0050          CALL    SCAN_KEYS       ;GET KEY HIT
000A 0F00            0051          XORLW   0               ;EXCL. OR WITH 0
000B 0743 0A08       0052          BNZ     CHK_FOR_KEY     ;KEY STILL PRESSED
                     0053                                  ;THEN LOOP
                     0054 NO_KEY_PRESSED
000D 0446            0055          BCF     PORT_B,SCAN1    ;SET SCAN LINES LOW
000E 0466            0056          BCF     PORT_B,SCAN2    ;       /
000F 0003            0057          SLEEP                   ;SLEEP
                     0058 ;
                     0060 ;
                     0061 INIT_PORT_B
0010 0C03            0062          MOVLW   B'00000011'     ; config RB0, 1 as i/p's
0011 0006            0063          TRIS    PORT_B          ;  and RB2-7 as o/p's
0012 0CFF            0064          MOVLW   0FFh
0013 0026            0065          MOVWF   PORT_B          ;DEFAULT VALUES FOR PORT_B
0014 0800            0066          RETLW   0               ;RETURN WITH NO ERROR
```

© 1993 Microchip Technology Incorporated

```
                        0067 ;
                        0068 ;This routine, scans two keys and returns the following:
                        0069 ;       0 if no key is pressed
                        0070 ;       1 if SW1 is pressd
                        0071 ;       2 if SW2 is pressed
                        0072 ;       3 if SW1 and SW2 are pressed
                        0073 ;
                        0074 SCAN_KEYS
0015 0446               0075         BCF     PORT_B,SCAN1    ;ENABLE SCAN FOR SW1
0016 0466               0076         BCF     PORT_B,SCAN2    ;EANBLE SCAN FOR SW2
0017 0C03               0077         MOVLW   B'00000011'     ;LOAD MASK IN W
0018 0146               0078         ANDWF   PORT_B,0        ;AND WITH PORT
0019 0546               0079         BSF     PORT_B,SCAN1    ;DISABLE SCAN
001A 0566               0080         BSF     PORT_B,SCAN2    ;       /
001B 01E2               0081         ADDWF   PC,1            ;GET OFFSET TO TABLE
001C 0803               0082         RETLW   3               ;SW1 AND SW2 PRESSED
001D 0802               0083         RETLW   2               ;SW2 PRESSED
001E 0801               0084         RETLW   1               ;SW1 PRESSED
001F 0800               0085         RETLW   0               ;NO KEY PRESSED
                        0086 ;
                        0087 ;DELAY, IS A APPROX. WAIT FOR 20.4mSECS, FOR A SYSTEM
                        0088 ;USING A 2 Mhz CRYSTAL CLOCK.
                        0089 DELAY
0020 0C14               0090         MOVLW   MSEC_20
0021 0028               0091         MOVWF   DB1
                        0092 DLY1
0022 0069               0093         CLRF    DB2
0023 02E8               0094         DECFSZ  DB1
0024 0A26               0095         GOTO    DLY2
0025 0800               0096         RETLW   0
                        0097 DLY2
0026 02E9               0098         DECFSZ  DB2             ;INNER LOOP = 1.02 MSEC.
0027 0A26               0099         GOTO    DLY2            ;       /
0028 0A22               0100         GOTO    DLY1
                        0101 ;
                        0102 ;
                        0103 TURN_GREEN_ON
0029 0486               0104         BCF     PORT_B,GRN_LED
002A 0800               0105         RETLW   0
                        0106 ;
                        0107 TURN_RED_ON
002B 04A6               0108         BCF     PORT_B,RED_LED
002C 0800               0109         RETLW   0
                        0110 ;
                        0111         END
                        0112
                        0113


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX-- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Incorporated

Interfacing to AC Power Lines

INTRODUCTION

This application note describes a simple method for measuring parameters from the AC power line. Parameters such as zero crossing, frequency, and relative phase can be measured. The method is useful for measurements on 50, 60, and 400 Hz power systems with voltages up to several hundred volts. The method requires only one external component, a resistor, and is more reliable than previously published methods using capacitors or bulky, expensive transformers.

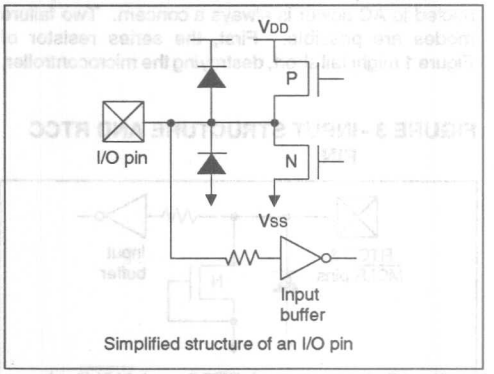
APPLICATIONS

This measurement method can be used in any application where power line parameters are used for system measurements or control. Typical applications are for switch timing (what part of the power cycle should the system be activated), power factor correction, power measurement, and power line monitor. An additional application is to generate timing or clock functions using the relatively stable power line frequency. The method is also useful for calibrating the oscillator frequency for accurate timing measurements when an inaccurate reference such as an RC oscillator is used to clock the PIC16C5X.

THEORY OF OPERATION

The application takes advantage of the input static protection circuitry that exists on all I/O pins of a CMOS PIC16C5X. These protection circuits are designed to short the inputs to the power supplies when a large overvoltage is applied, thus protecting the chip from static electricity spikes. On the PIC16C5X microcontrollers, this protection circuit is two large P-N diodes on each input (see Figure 1). These diodes will short any voltage higher than VDD to the VDD supply and any voltage less than Vss to the Vss supply. They can take several milliamps of current without any damage to the chip. High voltages can be applied directly to the chip inputs as long as they are current limited.

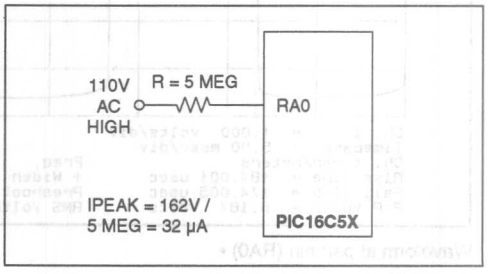
FIGURE 1 - PIC16C5X SERIES INPUT PROTECTION CIRCUIT ON I/O PINS



Simplified structure of an I/O pin

The least expensive method of current limiting is using a high value resistor. This method is shown schematically in Figure 2. The power line voltage is current limited by the resistor and then clamped by the input protection diodes internal to the PIC16C5X. A typical input waveform is shown in Figure 5. A 115V AC, 60 cycle sine wave will traverse from 0 to 2 volts in 32 µs so a typical threshold of 2 volts on the PIC16C5X I/O port will permit zero crossing detection accuracy of about 30 µs. If the typical capacitance on an I/O pin is 5 pF, then R should be (T = RC) 6 MEGohm or less for best zero crossing accuracy. A 5 MEGohm resistor with 115V AC applied to it will limit current to 32 µA, a value which is well within the safety margin of the PIC16C5X.

FIGURE 2 - CURRENT LIMITING USING AN EXTERNAL RESISTOR



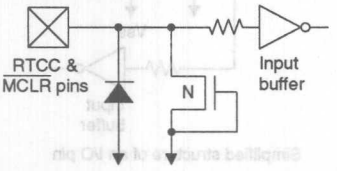
© 1993 Microchip Technology Inc.

The user needs to be aware that the circuit required to connect the RTCC input to AC power line is slightly different. Each of the I/O pins has two diodes for input protection whereas RTCC pin has only one protection diode connecting to Vss (see Figure 3). Therefore, it is necessary to connect a diode externally between RTCC pin and VDD in order to clamp the voltage on RTCC pin to VDD + 0.6V (approx.). See Figure 4. It is also recommended that resistor R be at least 2MΩ.

RELIABILITY

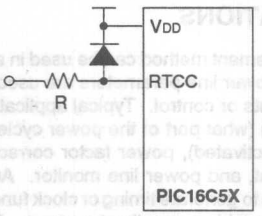
Reliability of production devices that are directly connected to AC power is always a concern. Two failure modes are possible. First, the series resistor of Figure 1 might fail short, destroying the microcontroller. This is the most unlikely failure mode of a resistor, and resistors are more reliable than transformers or capacitors, which are the alternate components for measuring line parameters. This reliability can be enhanced even further by using two resistors in series. Both would have to fail short to cause catastrophic failure, a very unlikely event. The second possible failure mode is that excessive injection of current into the PIC16C5X input might cause the protection diode to open. This would allow the input to go to the power line peak voltage (162V) and short the input transistor gate oxide, causing device failure. The maximum continuous injection current into an I/O pin is specified ±500 µA. An I/O pin is also capable of handling larger injection current (>100 mA) for a very short period (transient) of time. Therefore higher transient currents due to line voltage surges will be easily handled.

FIGURE 3 - INPUT STRUCTURE AND RTCC PIN



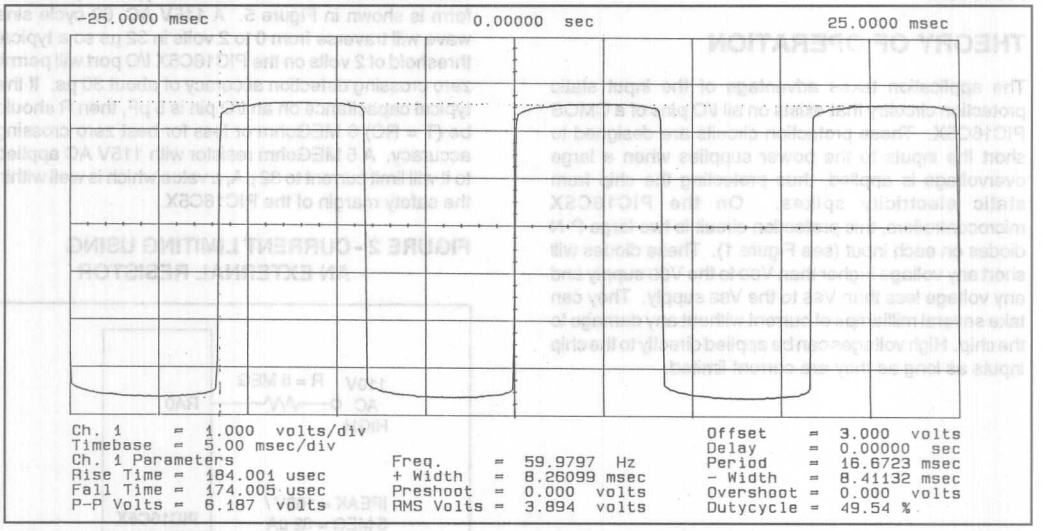
Simplified structure of RTCC and $\overline{MCLR}$ pins. Grounded gate NMOS device provides ESD and overvoltage protection.

FIGURE 4 - CONNECTING AC POWER LINE TO RTCC PIN



Connecting to RTCC input. R ≥ 2MΩ

FIGURE 5 - INPUT WAVEFORM



Waveform at part pin (RA0) •

R = 100K; Line: 60 Hz, 110 V

Author: Doug Cox
Logic Products Division

© 1993 Microchip Technology Inc.

AN592

Frequency Counter Using PIC16C5X

INTRODUCTION

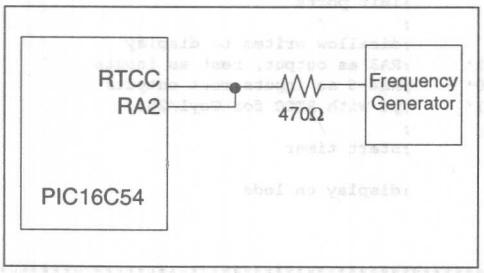
The PIC16C5X has one 8-bit timer (RTCC), which can be used with an 8-bit prescaler. The prescaler runs asynchronously, hence it can count a very high frequency. The minimum rise and fall times of the input frequency are specified to be 10nS, so the fastest clock rate the RTCC can count is 50 MHz. The prescaler must be used when measuring high frequency. Since the prescaler can be configured as a divide by 256 counter, the maximum resolution which the input frequency can be measured is 16 bits. However, the prescaler cannot be directly read like a file register. This application note depicts a unique method by which the user can "extract" the 8-bit value in the prescaler, whereby the resolution of the measurement is 16 bits with the high 8 bits in the RTCC and the low 8 bits in the prescaler.

IMPLEMENTATION

A frequency counter which can read frequencies from 50 MHz to 50 Hz was implemented in this application note to demonstrate this method of measuring the 16-bit counter value from the prescaler and RTCC.

The basic hardware for the measurement circuit is depicted in Figure 1. It consists of the frequency input at RTCC or RA4 (pin 3 in a PIC16C54). RA4 is connected to RA2. The input frequency is connected to RTCC through a 470 ohm resistor.

FIGURE 1



The RTCC is configured to measure the input frequency at RA4 of the PIC16C54. The input frequency is "gated" for a precise duration of time. Before starting this precise "gate", the RTCC is cleared (which also clears the prescaler), and the RA2 pin is configured as an input. The precise "gate" is implemented in software as an accurate delay. At the end of the delay, the RA2 pin is configured as an output going low. This will cause the input to the RTCC to be "halted" or "stopped". A 16-bit value of the input frequency is now saved in RTCC and the 8-bit prescaler. The high 8 bits are in RTCC and can be easily read. The low 8 bits have to be "shifted out". The 8-bits in the prescaler are "shifted out" by toggling RA2 with a "BSF" and "BCF" instruction. After every toggle, the value in RTCC is checked to see if the RTCC has incremented. If the number of toggles required to cause the RTCC to increment by 1 is N, then the 8-bit value in the pre-scaler can be calculated to be = (256 - N). By concatenating the calculated value and the original value in RTCC, the 16-bit value for the frequency is determined.

To measure a wide range of frequency, the following intermediate steps were taken:

Frequency Range	Precise "gate" Delay	Resolution
50 MHz - 10 MHz	1 ms	±10 KHz
10 MHz - 1 MHz	5 ms	±2 KHz
1 MHz - 100 KHz	50 ms	±200 Hz
100 KHz - 10 KHz	200 ms	±50 Hz
50 KHz - 50 Hz	50 ms†	±2 Hz

† In this case, the RTCC uses the internal 4 MHz clock and counts the number of instance of the external clock. Maximum Time required is 50 ms to make a ± 2 Hz accurate measurement for 10 KHz input frequency.

The check for the correct frequency is done automatically starting with the high frequency and going down to the low frequency. The maximum time required for each conversion is approximately 310 ms. In other words, three frequency checks are done every second.

CONCLUSION

The PIC16C5X family can be used to make a 16-bit measurement of input frequency with a small overhead of one resistor and one I/O port.

Author: Stan D'Souza
Logic Products Division

© 1994 Microchip Technology Inc.

APPENDIX A

```
MPASM 00.00.66 Beta              6-16-1994  23:59:52                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;Title DISPLAY.ASM
                     0002 ;This file displays a binary value found in the display
                     0003 ;register "DisplayRegister". The binary value is converted
                     0004 ;to BCD and then displayed.
                     0005         list p=16c71,f=inhx8m
                     0006         include "16cxx.h"
                     0179
                     0180
                     0181
                     0006
                     0007 ;
0001                 0008 TRUE    equ     1
0000                 0009 FALSE   equ     0
0000                 0010 FUZZY   equ     FALSE
                     0011         if FUZZY
                     0012         #define _ledEn  _portb,3
                     0013         #define _ledData _portb,1
                     0014         #define _ledClk _portb,2
                     0015         else
0046                 0016         #define _ledEn  _portb,3
0047                 0017         #define _ledData _portb,2
0048                 0018         #define _ledClk _portb,1
                     0019         endif
0011                 0020 HighFreq        equ     0x11
0012                 0021 LowFreq         equ     0x12
001A                 0022 acca    equ     1a
001B                 0023 accb    equ     1b
001C                 0024 accc    equ     1c
001D                 0025 accd    equ     1d
001E                 0026 acce    equ     1e
001F                 0027 accf    equ     1f
0009                 0028 time    equ     09
0010                 0029 temp    equ     10
                     0030 ;
                     0031 ;
                     0032         org     0
                     0033 start
0000 3004            0034         movlw   .4              ;initialize time
0001 0089            0035         movwf   time            ;       /
0002 019B            0036         clrf    accb            ;
0003 0186            0037         clrf    _portb          ;init ports
0004 0185            0038         clrf    _porta          ;       /
0005 1586            0039         bsf     _ledEn          ;disallow writes to display
0006 3017            0040         movlw   B'00010111'     ;RA3 as output, rest as inputs
0007 3070            0042         movlw   B'01110000'     ;RB4-6 as inputs rest outputs
0008 3087            0044         movlw   B'10000111'     ;ps with RTCC for Tcyl/256
0009 0062            0045         option                  ;       /
000A 0181            0046         clrf    _rtcc           ;start timer
                     0047 wait
000B 202D            0048         call    Display         ;display on leds
000C 280B            0049         goto    wait
                     0050 ;
                     0051
                     ;*****************************************************************
                     0052 ;       This subroutine converts a 8 bit binary word
                     0053 ;       into a 3 digit BCD
                     0054 ;       The input is in accb
                     0055 ;       output is in accc and accd with  lsd in ACCD.
                     0056 ;       The basic idea is that a 8 bit binary # has a value
                     0057 ;       between 0 and 255. First we check if the # is > 99
                     0058 ;       then if it is > 199. After each check we inc the MSD
                     0059 ;       Lastly we convert the LSD which will have a value
                     0060 ;       between 0 and 99.
```


© 1994 Microchip Technology Inc.

```
                    0061 ;
                    0062 ;*****************************************************************
                    0063 ;
                    0064 Bin8toBcd3
000D 3002           0065         movlw   2
000E 009C           0066         movwf   accc
000F 0190           0067         clrf    temp
0010 30C7           0068         movlw   .199        ;check if # is > 199
0011 021B           0069         subwf   accb,w      ;     /
0012 1903           0070         btfsc   _z          ;= 199?
0013 2816           0071         goto    Bcd99B
0014 1803           0072         btfsc   _c          ;     /
0015 2821           0073         goto    Bcd199      ;yes then do >200 #
                    0074 Bcd99B
0016 039C           0075         decf    accc        ;else inc Msd of BCD
0017 3063           0076         movlw   .99         ;and see > 99
0018 021B           0077         subwf   accb,w      ;     /
0019 1903           0078         btfsc   _z          ; == 99?
001A 281D           0079         goto    Bcd99A      ;yes then skip over
001B 1803           0080         btfsc   _c          ;     /
001C 2821           0081         goto    Bcd199      ;no then do 99
                    0082 Bcd99A
001D 039C           0083         decf    accc
                    0084 Bcd99
001E 081B           0085         movf    accb,w
001F 009D           0086         movwf   accd
0020 2823           0087         goto    get10th
                    0088 Bcd199
0021 009D           0089         movwf   accd        ;get result in ACCD
0022 039D           0090         decf    accd        ;dec to get correct value
                    0091 get10th
0023 300A           0092         movlw   .10
0024 021D           0093         subwf   accd,w      ;reduce by 10
0025 1C03           0094         btfss   _c          ;see if done
0026 282A           0095         goto    BcdOver     ;yes then end
0027 009D           0096         movwf   accd        ;get new value in ACCD
0028 0A90           0097         incf    temp        ;inc 10s count
0029 2823           0098         goto    get10th     ;do next
                    0099 BcdOver
002A 0E10           0100         swapf   temp,w      ;get in w
002B 049D           0101         iorwf   accd        ;or with 1s
002C 0008           0102         return
                    0103 ;
                    0104 ;*************************************************************
                    0105 ;       This routine displays 3 digits on a LT8522 display.
                    0106 ;       Three wires are required to drive the display
                    0107 ;       Enable -> active low when writing to display
                    0108 ;       Clock  -> 1 start followed by 35 more (36 total)
                    0109 ;                 36 clock required for load to occur.
                    0110 ;                 Rising edge of Clock is active.
                    0111 ;       Data   -> start data bit = high;
                    0112 ;                 1st data bit -> segment A of MSD
                    0113 ;                 2nd data bit -> segment B of MSD
                    0114 ;                 so on...
                    0115 ;                 8th data bit -> d.p. of MSD
                    0116 ;                 9th data bit -> segment A of 2nd digit
                    0117 ;                 10th data bit -> segment B of 2nd digit
                    0118 ;                 so on...
                    0119 ;                 16th data bit -> d.p. of 2nd digit
                    0120 ;                 17th data bit -> segment A of LSD
                    0121 ;                 18th data bit -> segment B of LSD
                    0122 ;                 so on ...
                    0123 ;                 24th data bit -> d.p. of LSD
                    0124 ;                 25th data bit -> appears on pin 4 of display
                    0125 ;                 26th data bit -> appears on pin 5 of display
                    0126 ;                 so on ...
                    0127 ;                 34th data bit -> appears on pin 13 of display.
                    0128 ;                 to dirve segment set data = high.
```

© 1994 Microchip Technology Inc.

```
                     0129 ;
                     0130 ;       The routine does a leading zero blanking.
                     0131 ;       The 3 BCD nibbles should be available in accc and accd,
                     0132 ;       with the MSD in the low nibble of accc.
                     0133 ;*****************************************************************
                     0134 Display
002D 205E            0135         call    StartDisplay
002E 0E11            0136         swapf   HighFreq,w
002F 390F            0137         andlw   0x0f
0030 2040            0138         call    LedValue
0031 2052            0139         call    DisplayW
0032 0811            0140         movf    HighFreq,w
0033 390F            0141         andlw   0x0f
0034 2040            0142         call    LedValue
0035 2052            0143         call    DisplayW
0036 0E12            0144         swapf   LowFreq,w
0037 390F            0145         andlw   0x0f
0038 2040            0146         call    LedValue
0039 2052            0147         call    DisplayW
003A 0812            0148         movf    LowFreq,w
003B 390F            0149         andlw   0x0f
003C 2040            0150         call    LedValue
003D 2052            0151         call    DisplayW
003E 2064            0152         call    EndDisplay
003F 3400            0153         retlw   0
                     0154 ;
                     0155 ;
                     0156 ;
                     0157 LedValueAddress
                     0158         if LedValueAddress < 0x100
                     0159 LedValue
0040 018A            0160         clrf    _pclath
0041 0782            0161         addwf   _pcl
0042 34FC            0162         retlw   0xfc    ;code for 0
0043 3460            0163         retlw   0x60    ;code for 1
0044 34DA            0164         retlw   0xda    ;code for 2
0045 34F2            0165         retlw   0xf2    ;code for 3
0046 3466            0166         retlw   0x66    ;code for 4
0047 34B6            0167         retlw   0xb6    ;code for 5
0048 34BE            0168         retlw   0xbe    ;code for 6
0049 34E0            0169         retlw   0xe0    ;code for 7
004A 34FE            0170         retlw   0xfe    ;code for 8
004B 34E6            0171         retlw   0xe6    ;code for 9
004C 34EE            0172         retlw   0xee    ;code for A
004D 343E            0173         retlw   0x3e    ;code for b
004E 349C            0174         retlw   0x9c    ;code for C
004F 347A            0175         retlw   0x7a    ;code for d
0050 349E            0176         retlw   0x9e    ;code for E
0051 348E            0177         retlw   0x8e    ;code for F
                     0178         endif
                     0179 ;
                     0180 ;
                     0181 DisplayW
0052 0090            0182         movwf   temp
0053 3008            0183         movlw   .8
0054 009A            0184         movwf   acca
                     0185 DisplayLoop
0055 0D90            0186         rlf     temp
0056 1803            0187         btfsc   _c
0057 1506            0188         bsf     _ledData
0058 1486            0189         bsf     _ledClk
0059 1086            0190         bcf     _ledClk
005A 1106            0191         bcf     _ledData
005B 0B9A            0192         decfsz  acca
005C 2855            0193         goto    DisplayLoop
005D 3400            0194         retlw   0
                     0195 ;
                     0196 ;
```


© 1994 Microchip Technology Inc.

```
                        0197 StartDisplay
005E 1186               0198         bcf     _ledEn
005F 1506               0199         bsf     _ledData
0060 1486               0200         bsf     _ledClk
0061 1086               0201         bcf     _ledClk
0062 1106               0202         bcf     _ledData
0063 3400               0203         retlw   0
                        0204 ;
                        0205 EndDisplay
0064 1486               0206         bsf     _ledClk
0065 1086               0207         bcf     _ledClk
0066 1486               0208         bsf     _ledClk
0067 1086               0209         bcf     _ledClk
0068 1486               0210         bsf     _ledClk
0069 1086               0211         bcf     _ledClk
006A 1586               0212         bsf     _ledEn
006B 3400               0213         retlw   0
                        0214
                        0215 ;
                        0216         org     0x1ff
01FF 2800               0217         goto    start
                        0218
                        0219 ;
                        0220         end
                        0221 ;
                        0222
                        0223


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```

NOTES:

© 1994 Microchip Technology Inc.

AN513

Analog to Digital Conversion

INTRODUCTION

This application note describes a method for implementing analog to digital conversion on the PIC16C5X series of microcontrollers. The converter requires only five external components and is software and hardware configurable for conversion resolutions from 6 bits up to 10 bits and conversion times of 250µs or longer. The method is useable for both voltage and current conversion and uses a software calibration technique that compensates for time and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X microcontrollers are offered as One-Time-Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X's Sleep mode.
* PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (see Figure 1) to convert the input voltage to time, which can be easily measured using a microcontroller. First, the reference voltage is applied to the input voltage to current converter (U1). The equivalent circuit is shown in Figure 2. This circuit provides a linearly variable current as a function of input voltage. The logarithmic characteristic that would occur if the input voltage was applied directly to an RC is not present. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the resistor and capacitor, changes in the input threshold voltage and temperature variations. After the software calibration value is measured, the capacitor is discharged (see Figure 3) and the input voltage is connected to VIN. The time to trip the threshold is measured for the input voltage and compared to the calibration value to determine the actual input voltage.

FIGURE 1 - VOLTMETER A TO D CONVERTER

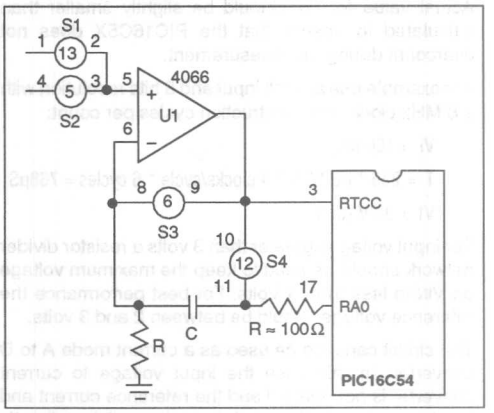


FIGURE 2 - VOLTMETER MEASUREMENT CYCLE

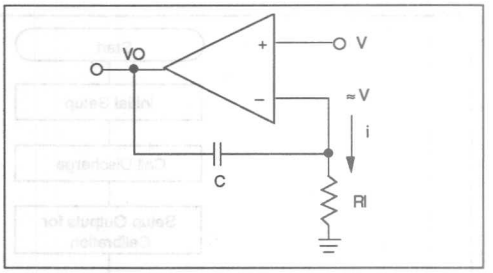
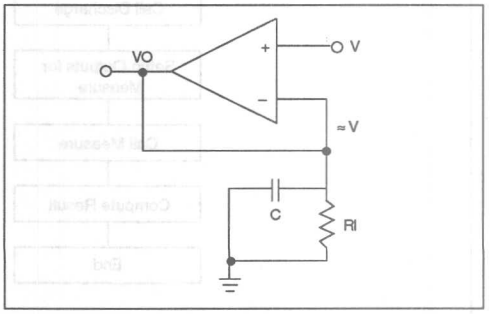


FIGURE 3 - VOLTMETER DISCHARGE CYCLE



© 1994 Microchip Technology Inc.

CIRCUIT CONFIGURATION

The values of R and C are selected based upon the number of bits of resolution required.

RC = (Vi . T)/Vt

Where:

Vi = Lowest voltage to be measured (at least ten lsb's)

T = Time to do the number of bits of resolution desired

Vt = Threshold voltage of the PIC16C5X input being used

Actual value for RC should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use a 3 volt input and 8 bits resolution with a 8 MHz clock and 6 instruction cycles per count:

Vi = 100 mV

T = 256 *1/8 MHz * 4 clocks/cycle * 6 cycles = 768µS

Vt = 3.0V (est)

For input voltages greater than 3 volts a resistor divider network should be used to keep the maximum voltage on VIN to less than 3 Volts. For best performance the reference voltage should be between 2 and 3 volts.

The circuit can also be used as a current mode A to D converter. In this case the input voltage to current converter is not needed and the reference current and input current are both routed via analog switches directly into the capacitor.

CIRCUIT PERFORMANCE

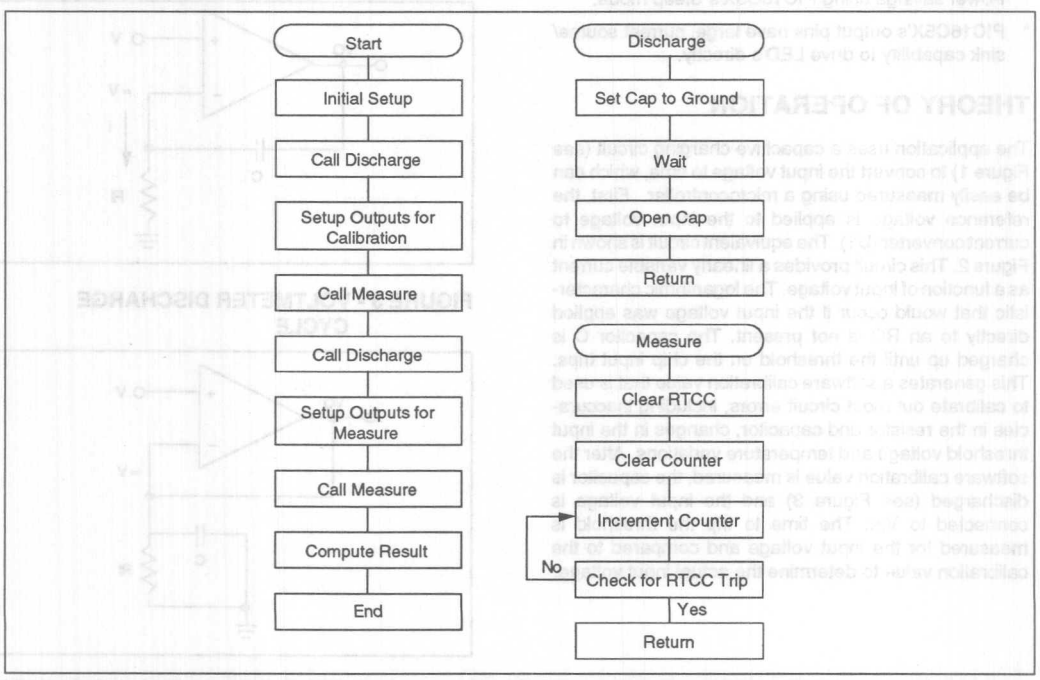
The calibration cycle removes all first order errors (offset, gain, R and C inaccuracy, power supply voltage and temperature) except the reference voltage drift. Any change in the reference voltage, including noise, between the calibration cycle and the measurement cycle may result in measurement errors. Other error sources are analog switch leakage, resistor and capacitor nonlinearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the converter to be accurate within +/- 1% of full scale.

Example

Assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures the time up to 16 bits and calculates the results using 16-bit multiply and divide subroutines. In actual applications, if measurement accuracy permits, it may be advantageous to use 8 bits. The math code can be substantially reduced and the measure time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 4 - TRANSMISSION FLOW CHART



© 1994 Microchip Technology Inc.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1
VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90



                      TITLE  'VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90'
                      LIST   P=16C54,F=inhx16,n=0

0008                  ACCA    EQU     8
000A                  ACCB    EQU     0A
000C                  ACCC    EQU     0C
000E                  ACCD    EQU     0E
0010                  ACCE    EQU     10
0012                  TMEAS   EQU     12
0014                  TEMP    EQU     14

0060                  VCALMS  EQU     60              ;VCAL MSB VALUE IN HEX
00A4                  VCALLS  EQU     0A4             ;VCAL LSB VALUE IN HEX

                      ORG 1FF
01FF 0A58             GOTO    VOLTS           ;PROGRAM CODE
                      ORG     0               ;SUBROUTINES

0000 0209             MADD    MOVF    ACCA+1,W
0001 01EB                     ADDWF   ACCB+1          ;ADD LSB
0002 0603                     BTFSC   3,0             ;ADD IN CARRY
0003 02AA                     INCF    ACCB
0004 0208                     MOVF    ACCA,W
0005 01EA                     ADDWF   ACCB            ;ADD MSB
0006 0800                     RETLW   0
0007 0000                     NOP

0008 0915             MPY     CALL    SETUP           ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E             MLOOP   RRF     ACCD            ;ROTATE D RIGHT
000A 032F                     RRF     ACCD+1
000B 0603                     SKPNC                   ;NEED TO ADD?
000C 0900                     CALL    MADD
000D 032A                     RRF     ACCB
000E 032B                     RRF     ACCB+1
000F 032C                     RRF     ACCC
0010 032D                     RRF     ACCC+1
0011 02F4                     DECFSZ  TEMP            ;LOOP UNTIL ALL BITS CHECKED
0012 0A09                     GOTO    MLOOP
0013 0800                     RETLW   0

0014 0000                     NOP
0015 0C10             SETUP   MOVLW   10
0016 0034                     MOVWF   TEMP
0017 020A                     MOVF    ACCB,W          ;MOVE B TO D
0018 002E                     MOVWF   ACCD
0019 020B                     MOVF    ACCB+1,W
001A 002F                     MOVWF   ACCD+1
001B 020C                     MOVF    ACCC,W
001C 0030                     MOVWF   ACCE
001D 020D                     MOVF    ACCC+1,W
001E 0031                     MOVWF   ACCE+1
001F 006A                     CLRF    ACCB
0020 006B                     CLRF    ACCB+1
0021 0800                     RETLW   0

0022 0000                     NOP
0023 0915             DIV     CALL    SETUP
0024 0C20                     MOVLW   20
0025 0034                     MOVWF   TEMP
0026 006C                     CLRF    ACCC
0027 006D                     CLRF    ACCC+1
```

© 1994 Microchip Technology Inc.

```
0028 0403          DLOOP   CLRC
0029 0371                  RLF     ACCE+1
002A 0370                  RLF     ACCE
002B 036F                  RLF     ACCD+1
002C 036E                  RLF     ACCD
002D 036D                  RLF     ACCC+1
002E 036C                  RLF     ACCC
002F 0208                  MOVF    ACCA,W
0030 008C                  SUBWF   ACCC,W          ;CHECK IF A>C
0031 0743                  SKPZ
0032 0A35                  GOTO    NOCHK
0033 0209                  MOVF    ACCA+1,W
0034 008D                  SUBWF   ACCC+1,W        ;IF MSB EQUAL THEN CHECK LSB
0035 0703          NOCHK   SKPC                    ;CARRY SET IF C>A
0036 0A3E                  GOTO    NOGO
0037 0209                  MOVF    ACCA+1,W        ;C-A INTO C
0038 00AD                  SUBWF   ACCC+1
0039 0703                  BTFSS   3,0
003A 00EC                  DECF    ACCC
003B 0208                  MOVF    ACCA,W
003C 00AC                  SUBWF   ACCC
003D 0503                  SETC                    ;SHIFT A 1 INTO B (RESULT)
003E 036B          NOGO    RLF     ACCB+1
003F 036A                  RLF     ACCB
0040 02F4                  DECFSZ  TEMP            ;LOOP UNTILL ALL BITS CHECKED
0041 0A28                  GOTO    DLOOP
0042 0800                  RETLW   0

0043 0C0E          DSCHRG  MOVLW   B'00001110'     ;DISCHARGE C (RA0 ON)
0044 0005                  TRIS    5
0045 0CFF                  MOVLW   0FF
0046 0034                  MOVWF   TEMP
0047 02F4          LOOP    DECFSZ  TEMP            ;WAIT
0048 0A47                  GOTO    LOOP
0049 0C0F                  MOVLW   B'00001111'     ;ALL RA HIGH Z
004A 0005                  TRIS    5
004B 0800                  RETLW   0

004C 0061          M_TIME  CLRF    1               ;CLEAR RTCC REGISTER
004D 0069                  CLRF    ACCA+1          ;CLEAR 16 BIT COUNTER
004E 0068                  CLRF    ACCA
004F 03E9          TLOOP   INCFSZ  ACCA+1
0050 0A54                  GOTO    ENDCHK
0051 03E8                  INCFSZ  ACCA
0052 0A54                  GOTO    ENDCHK
0053 0A56                  GOTO    END_M
0054 0701          ENDCHK  BTFSS   1,0             ;CHECK FOR RTCC TRIP
0055 0A4F                  GOTO    TLOOP
0056 0201          END_M   MOVF    1,W
0057 0800                  RETLW   0

0058 0C06          VOLTS   MOVLW   B'00000110'     ;SET S2 AND S3 HIGH(ON WHEN ACTIVATED)
0059 0026                  MOVWF   6
005A 0CF0                  MOVLW   B'11110000'     ;ACTIVATE SWITCHES S1-S4
005B 0006                  TRIS    6
005C 0C28                  MOVLW   B'00101000'     ;SELECT POSITIVE EDGE FOR RTCC
005D 0002                  OPTION
005E 0C00                  MOVLW   B'00000000'
005F 0025                  MOVWF   5               ;SET RA0 LOW (ON WHEN ACTIVATED)

0060 0943          MEAS    CALL    DSCHRG          ;CHARGE CAPACITOR TO VIN
0061 0C0A                  MOVLW   B'00001010'     ;S2 AND S4 ON
0062 0026                  MOVWF   6
0063 094C                  CALL    M_TIME          ;MEASURE TIME
0064 0209                  MOVF    ACCA+1,W
0065 0033                  MOVWF   TMEAS+1         ;STORE LSB
0066 0208                  MOVF    ACCA,W
0067 0032                  MOVWF   TMEAS           ;STORE MSB
```


© 1994 Microchip Technology Inc.

```
0068 0C05           CAL     MOVLW   B'00000101'     ;S1 AND S3 ON
0069 0026                   MOVWF   6
006A 0943                   CALL    DSCHRG          ;CHARGE CAPACITOR TO VREF
006B 0C09                   MOVLW   B'00001001'     ;S1 AND S4 ON
006C 0026                   MOVWF   6
006D 094C                   CALL    M_TIME          ;MEASURE TIME

006E 0CA4                   MOVLW   VCALLS
006F 002B                   MOVWF   ACCB+1
0070 0C60                   MOVLW   VCALMS
0071 002A                   MOVWF   ACCB

0072 0908                   CALL    MPY             ;MULTIPLY ACCA(TCAL) * ACCB(VREF)
0073 0213                   MOVF    TMEAS+1,W
0074 0029                   MOVWF   ACCA+1
0075 0212                   MOVF    TMEAS,W
0076 0028                   MOVWF   ACCA

0077 0923                   CALL    DIV             ;DIVIDE ACCB(TCAL * V) BY ACCA(TMEAS)

0078 0A58                   GOTO VOLTS

                            END

Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Implementing Ohmeter/Temperature Sensor

INTRODUCTION

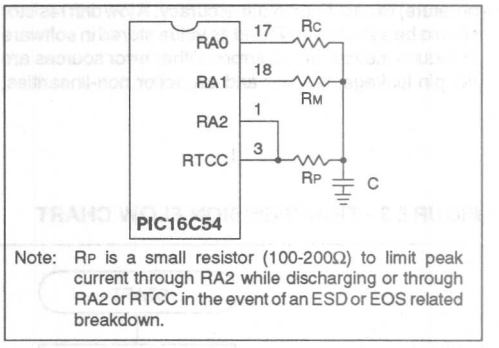
This application note describes a method for implementing an ohmmeter or resistance type temperature sensor using the PIC16C5X series of microcontrollers. The ohmmeter requires only two external components and is software and hardware configurable for resistance measurement with resolutions from 6 bits up to 10 bits with measurement times of 250 µs (6 bits at 8 MHz) or longer. The method uses a software calibration technique that compensates for voltage, time, and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X's Sleep mode.
* PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

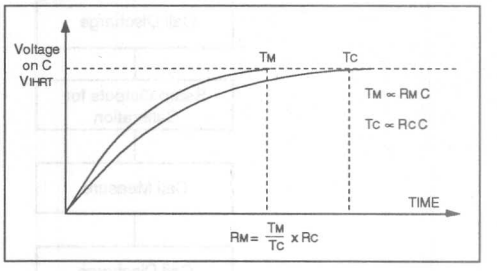
The application uses a capacitive charging circuit (Figure 1) to convert the resistance to time, which can be easily measured using a microcontroller. First, a reference voltage (usually VDD) is applied to a calibration resistor, Rc. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the capacitor, changes in the input threshold voltage and temperature variations. After C is discharged, the reference voltage is applied to the resistance to be measured (or thermistor). The time to trip the threshold is then measured and compared to the calibration value to determine the actual resistance (Figure 2). In the temperature sensing mode, the temperature is calculated using a lookup table.

FIGURE 1 - OHMETER/TEMPERATURE SENSOR



Note: RP is a small resistor (100-200Ω) to limit peak current through RA2 while discharging or through RA2 or RTCC in the event of an ESD or EOS related breakdown.

FIGURE 2 - OHMETER/TEMPERATURE SENSOR



CIRCUIT CONFIGURATION

The values of Rc and C are selected based upon the number of bits of resolution required. Rc should be approximately one half the largest value resistance to be measured and:

$$C = \frac{-T}{R_M * \ln\left(1 - \frac{Vt}{Vr}\right)}$$

Where:

Vr = Reference voltage

T = Time to do the number of bits of resolution desired

Vt = Threshold voltage of the PIC16C5X input being used

RM = Maximum resistance value to be measured

Actual value for C should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

© 1994 Microchip Technology Inc.

For example use RM=200K for 8-bits resolution with an 8 MHz clock, Vr = 5V, Vt = 3V, Rc =100K and 6 instruction cycles per count:

T = 256 counts * 1/8 MHz * 4 clocks/instruction * 6 instructions/count = 768 µs

C = 4200 pF [Use 3900 pF]

CIRCUIT PERFORMANCE

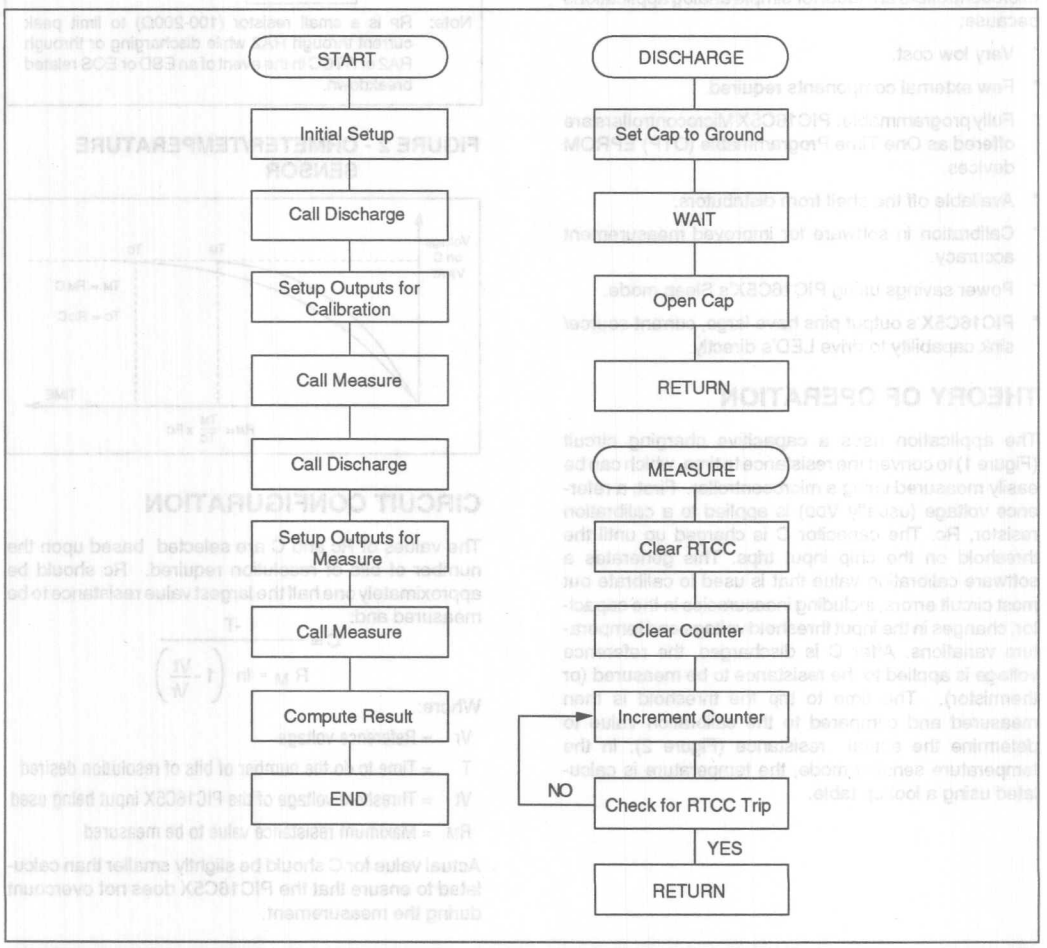
The calibration cycle removes all first order errors (offset, gain, C inaccuracy, power supply voltage and temperature) except R absolute accuracy. A low drift resistor should be selected for R and its value stored in software to reduce measurement errors. Other error sources are I/O pin leakage, resistor and capacitor non-linearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the ohmmeter to be accurate within +/- 1% over one decade.

Example

The assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures time up to 16 bits (65535 measure cycles) and calculates the results using 16 bit multiply and divide subroutines. In actual applications, it is more efficient to use 8 bit measurements if application accuracies permit. The math code will be substantially reduced and measurement time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 3 - TRANSMISSION FLOW CHART



© 1994 Microchip Technology Inc.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1


                        LIST   P=16C54,F=inhx8M

0008                ACCA    EQU    8
000A                ACCB    EQU    0A
000C                ACCC    EQU    0C
000E                ACCD    EQU    0E
0010                ACCE    EQU    10
0012                TCAL    EQU    12
0014                TEMP    EQU    14

002F                RCALMS  EQU    2F          ;RCAL MSB VALUE IN HEX
003C                RCALLS  EQU    3C          ;RCAL LSB VALUE IN HEX

                        ORG    1FF
01FF 0A58               GOTO   OHMS
                        ORG    0

0000 0209           MADD    MOVF   ACCA+1,W
0001 01EB                   ADDWF  ACCB+1      ;ADD LSB
0002 0603                   BTFSC  3,0         ;ADD IN CARRY
0003 02AA                   INCF   ACCB
0004 0208                   MOVF   ACCA,W
0005 01EA                   ADDWF  ACCB        ;ADD MSB
0006 0800                   RETLW  0
0007 0000                   NOP

0008 0915           MPY     CALL   SETUP       ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E           MLOOP   RRF    ACCD        ;ROTATE D RIGHT
000A 032F                   RRF    ACCD+1
000B 0603                   SKPNC              ;NEED TO ADD?
000C 0900                   CALL   MADD
000D 032A                   RRF    ACCB
000E 032B                   RRF    ACCB+1
000F 032C                   RRF    ACCC
0010 032D                   RRF    ACCC+1
0011 02F4                   DECFSZ TEMP        ;LOOP UNTIL ALL BITS CHECKED
0012 0A09                   GOTO   MLOOP
0013 0800                   RETLW  0

0014 0000                   NOP
0015 0C10           SETUP   MOVLW  10
0016 0034                   MOVWF  TEMP
0017 020A                   MOVF   ACCB,W      ;MOVE B TO D
0018 002E                   MOVWF  ACCD
0019 020B                   MOVF   ACCB+1,W
001A 002F                   MOVWF  ACCD+1
001B 020C                   MOVF   ACCC,W
001C 0030                   MOVWF  ACCE
001D 020D                   MOVF   ACCC+1,W
001E 0031                   MOVWF  ACCE+1
001F 006A                   CLRF   ACCB
0020 006B                   CLRF   ACCB+1
0021 0800                   RETLW  0
0022 0000                   NOP
0023 0915           DIV     CALL   SETUP
0024 0C20                   MOVLW  20
0025 0034                   MOVWF  TEMP
0026 006C                   CLRF   ACCC
0027 006D                   CLRF   ACCC+1
0028 0403           DLOOP   CLRC
0029 0371                   RLF    ACCE+1
002A 0370                   RLF    ACCE
002B 036F                   RLF    ACCD+1
```

© 1994 Microchip Technology Inc.

```
002C 036E                RLF     ACCD
002D 036D                RLF     ACCC+1
002E 036C                RLF     ACCC
002F 0208                MOVF    ACCA,W
0030 008C                SUBWF   ACCC,W          ;CHECK IF A>C
0031 0743                SKPZ
0032 0A35                GOTO    NOCHK
0033 0209                MOVF    ACCA+1,W
0034 008D                SUBWF   ACCC+1,W        ;IF MSB EQUAL THEN CHECK LSB
0035 0703          NOCHK SKPC                    ;CARRY SET IF C>A
0036 0A3E                GOTO    NOGO
0037 0209                MOVF    ACCA+1,W        ;C-A INTO C
0038 00AD                SUBWF   ACCC+1
0039 0703                BTFSS   3,0
003A 00EC                DECF    ACCC
003B 0208                MOVF    ACCA,W
003C 00AC                SUBWF   ACCC
003D 0503                SETC                    ;SHIFT A 1 INTO B (RESULT)
003E 036B          NOGO  RLF     ACCB+1
003F 036A                RLF     ACCB
0040 02F4                DECFSZ  TEMP            ;LOOP UNTILL ALL BITS CHECKED
0041 0A28                GOTO    DLOOP
0042 0800                RETLW   0

0043 0C0B          DSCHRG MOVLW  B'00001011'     ;ACTIVATE RA2
0044 0005                TRIS    5
0045 0CFF                MOVLW   0FF
0046 0034                MOVWF   TEMP
0047 02F4          LOOP  DECFSZ  TEMP            ;WAIT
0048 0A47                GOTO    LOOP
0049 0C0F                MOVLW   B'00001111'     ;ALL OUTPUTS OFF
004A 0005                TRIS    5
004B 0800                RETLW   0

004C 0061          M_TIME CLRF   1               ;CLEAR RTCC
004D 0069                CLRF    ACCA+1
004E 0068                CLRF    ACCA
004F 03E9          TLOOP INCFSZ  ACCA+1
0050 0A54                GOTO    ENDCHK
0051 03E8                INCFSZ  ACCA
0052 0A54                GOTO    ENDCHK
0053 0A56                GOTO    END_M
0054 0701          ENDCHK BTFSS  1,0             ;CHECK FOR RTCC TRIP
0055 0A4F                GOTO    TLOOP
0056 0201          END_M MOVF    1,W
0057 0800                RETLW   0

0058 0C03          OHMS  MOVLW   B'00000011'     ;SET RA0 AND RA1 HIGH (ON WHEN ACTIVATED)
0059 0025                MOVWF   5
005A 0C28                MOVLW   B'00101000'     ;SELECT POSITIVE EDGE FOR RTCC
005B 0002                OPTION

005C 0943          CAL   CALL    DSCHRG          ;DISCHARGE CAPACITOR
005D 0C0E                MOVLW   B'00001110'     ;ACTIVATE RA0
005E 0005                TRIS    5
005F 094C                CALL    M_TIME          ;MEASURE TIME
0060 0209                MOVF    ACCA+1,W
0061 0033                MOVWF   TCAL+1          ;STORE LSB
0062 0208                MOVF    ACCA,W
0063 0032                MOVWF   TCAL            ;STORE MSB

0064 0943          MEAS  CALL    DSCHRG          ;DISCHARGE CAPACITOR
0065 0C0D                MOVLW   B'00001101'     ;ACTIVATE RA1
0066 0005                TRIS    5
0067 094C                CALL    M_TIME          ;MEASURE TIME

0068 0C3C                MOVLW   RCALLS          ;CALIBRATION LSB VALUE
0069 002B                MOVWF   ACCB+1
006A 0C2F                MOVLW   RCALMS          ;CALIBRATION MSB VALUE
```


© 1994 Microchip Technology Inc.

```
006B 002A                MOVWF   ACCB

006C 0908                CALL    MPY             ;MULTIPLY ACCA(MEAS) * ACCB(RCAL)
006D 0213                MOVF    TCAL+1,W
006E 0029                MOVWF   ACCA+1
006F 0212                MOVF    TCAL,W
0070 0028                MOVWF   ACCA

0071 0923                CALL    DIV             ;DIVIDE ACCB(MEAS * R) BY ACCA(TCAL)

0072 0A58                GOTO    OHMS

                         END


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Implementing a Simple Serial Mouse Controller

INTRODUCTION

The mouse is becoming increasingly popular as a standard pointing data entry device. It is no doubt that the demand of the mouse is increasing. Various kinds of mice can be found in the market, including optical mouse, opto-mechanical mouse, and its close relative, trackball. The mouse interfaces to the host via an RS-232 port or a dedicated interface card. Their mechanisms are very similar. The major electrical components of a mouse are:

- Microcontroller
- Photo-transistors
- Infrared emitting diode
- Voltage conversion circuit

The intelligence of the mouse is provided by the microcontroller, hence the features and performance of a mouse is greatly related to the microcontroller used.

This application note describes the implementation of a serial mouse using the PIC16C54. The PIC16C54 is a high speed 8-bit CMOS microcontroller offered by Microchip Technology Inc. It is an ideal candidate for a mouse controller.

THEORY OF OPERATION

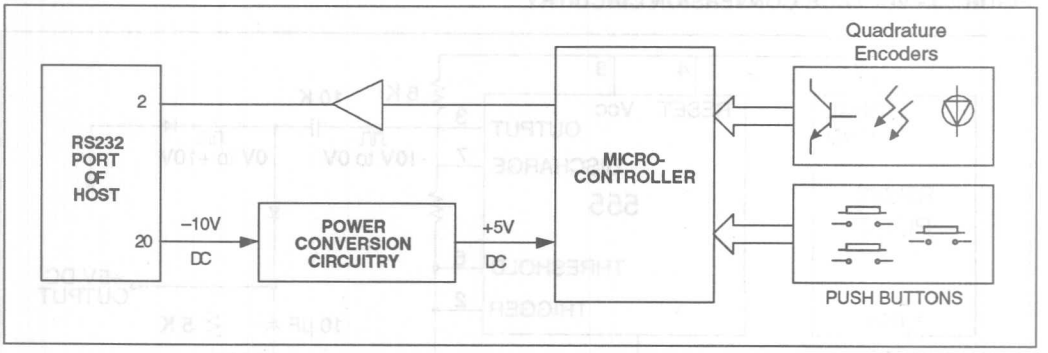
A mouse can be divided into several functional blocks:

- Microcontroller
- Button detection
- Motion detection
- RS-232 signal generation
- 5V DC power supply unit

A typical functional block diagram is shown in Figure 1.

In Figure 2, three push buttons are connected to the input ports of the PIC16C54. When a switch opening or closure is detected, a message is formatted and sent to the host. The X and Y movements are measured by counting the pulses generated by the photo-couplers. In the case of an opto-mechanical mouse, the infrared light emitted by the infrared diode is blocked by the rotating wheel, so that the pulses are generated on the photo-transistor side. In case of an optical mouse, the infrared light emitted by the infrared diode is reflected off the reflective pad patterned with vertical and horizontal grid lines. It is then received by the photo-transistor in the mouse. When any X or Y movement is detected, a message is formatted and sent to the host.

FIGURE 1 - FUNCTIONAL BLOCKS OF A SERIAL MOUSE



© 1993 Microchip Technology Inc.

The Microsoft® Mouse System and the Mouse Systems® device both use serial input techniques. The Mouse System protocol format contains five bytes of data. One byte describes the status of three push buttons, two bytes for the relative X movements and two bytes for the relative Y movements. The Microsoft protocol format contains three bytes of data describing the status of two push buttons and the relative X and Y movements. The details of these protocols are given in Table 1.

Three lines are connected to the host via the RS-232 port:

- Signal Ground
- Received Data
- Request to Send

"Received Data" carries the message sent by the mouse. While "Request to Send" provides a –10V DC for voltage conversion circuitry. A voltage of +5V DC is required for electronic components inside the mouse, however, +5V DC is not part of an RS-232 port, so voltage conversion circuitry is required. This circuit is typically composed of a 555 timer, Zener diodes, and capacitors. An example circuit is shown in Figure 3. Since the current supplied through the RS-232 port is limited to 10 mA, the mouse cannot be designed to consume more than 10 mA current unless an external power supply is provided. The PIC16C54, running at 4 MHz (1 µs instruction cycle) can provide a very high tracking speed. An 8 MHz version of PIC16C54 is also available if higher performance is desired.

FIGURE 2 - PIC16C54 PIN ASSIGNMENT

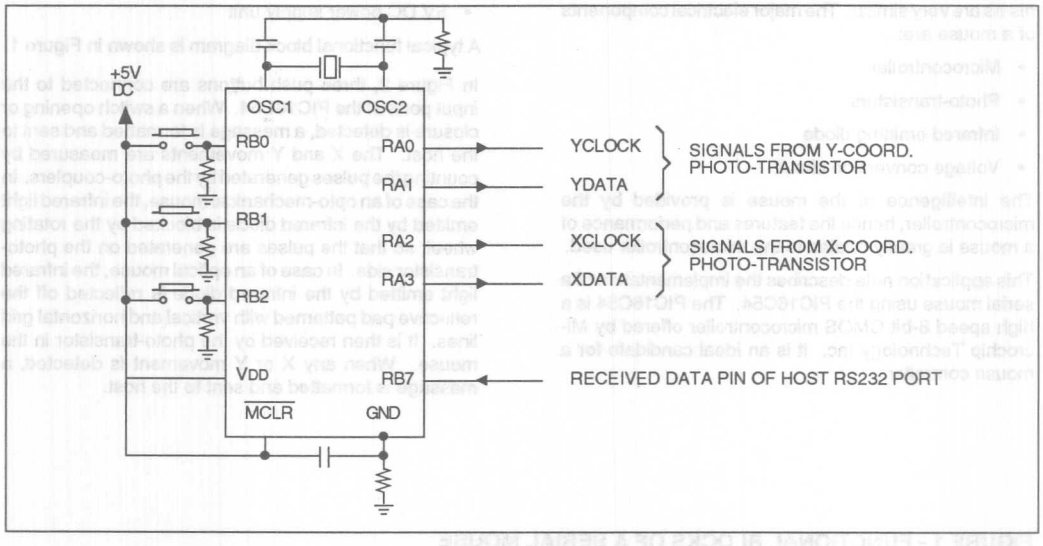
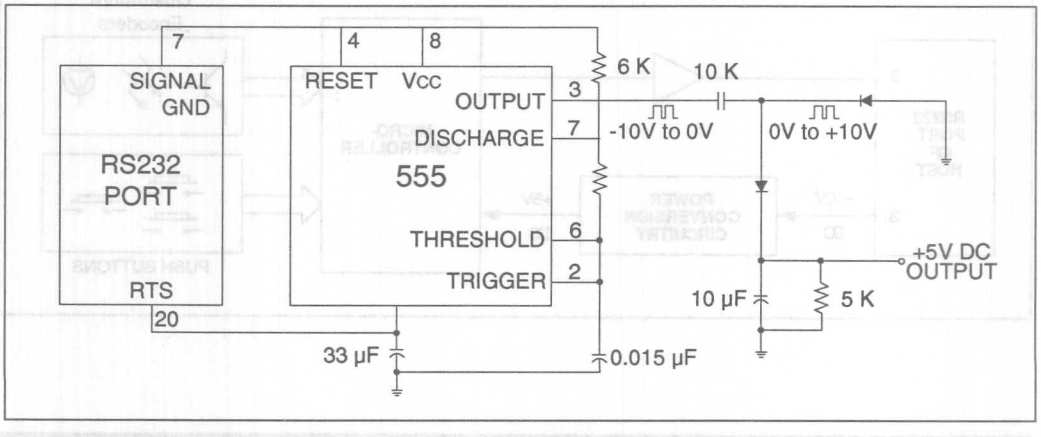


FIGURE 3 - VOLTAGE CONVERSION CIRCUITRY



© 1993 Microchip Technology Inc.

ABOUT THE SOFTWARE

The major tasks performed by the software are button scanning, X and Y motion scanning, formatting and sending serial data to the host. These tasks need to be performed in parallel in order to gain better tracking speed. The pulses generated by the photo-couplers are counted while transmitting the serial signals to the RS-232 port. The number of pulses reflects the speed of the movement. The more number of pulses, the faster the movement is.

The directions of the movement are determined by the last states and the present states of the outputs of the photo-transistors. In Figure 4, XCLOCK and XDATA are outputs from the photo-transistors corresponding to the X-axis movement. XDATA is read when a rising or a falling edge of XCLOCK is detected. For right movement, XDATA is either LOW at the rising edge of XCLOCK or HIGH at the falling edge of XCLOCK. The up and down movement detections follow the same logic. In Table 1, X7:X0 are data for relative movement. If X is positive, it implies that the mouse is moving to the right. If X is negative, it implies a movement to the left. Similarly, if Y is positive, it indicates that the mouse is moving down and if Y is negative, it indicates that the mouse is moving up. The pulses generated by the photo-couplers are checked before every bit is sent. A bit takes 1/1200 second to send, if the distance between the grid lines is 1 mm, the tracking speed will be up to 1200 mm/second.

2

FIGURE 4 - VOLTAGE CONVERSION CIRCUITRY

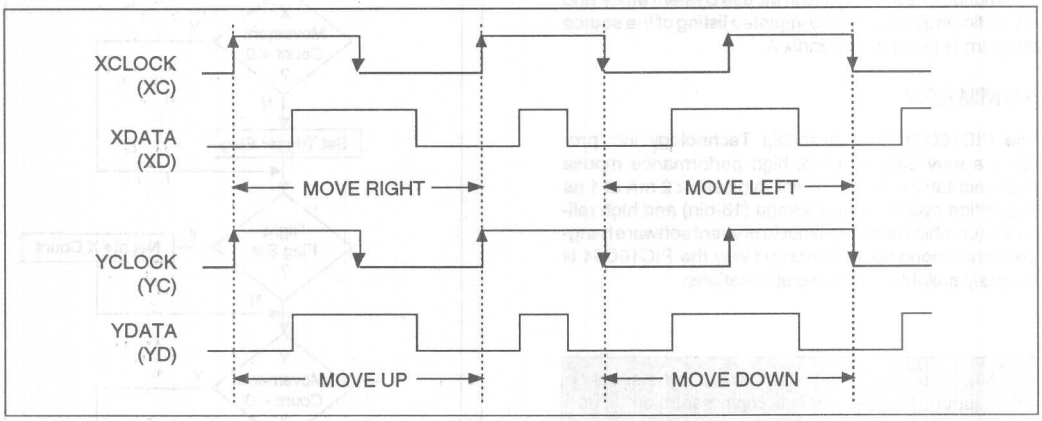


TABLE 1 - MOUSE SYSTEM AND MICROSOFT PROTOCOLS

	Mouse System Format*								Microsoft Format*							
Bit Position	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R	1	1	L	R	Y7	Y6	X7	X6
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0	0	0	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0	0	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0								
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0								

* L = Left Key Status
M = Middle Key Status
R = Right Key Status

1 = Pressed
0 = Released

X7-X0 = X-Axis Movement Data
Y7-Y0 = Y-Axis Movement Data

© 1993 Microchip Technology Inc.

The buttons are scanned after a message is sent and the time used to send the message is used as the debouncing time. The message is in an RS-232 format with 1200 baud, eight data bits, no parity, and two stop bits.

The flow charts of the main program, subroutine BYTE and subroutine BIT are shown in Figures 5, 6, and 7. Figure 5 shows the Trigger Flag is set when any change of button status or X/Y movement is detected. Subroutine BYTE is called in the main program five times to send five bytes of information. Subroutine BYTE controls the status of the "Received Data" (RD) pin. If Trigger Flag is clear, RD will always be HIGH. Hence, no message will be sent even when subroutine BYTE is called. Figure 7 shows that subroutine BIT counts the number of pulses from outputs of the photo-transistors, determines the directions, and generates 1/1200 second delay to get 1200 baud timing.

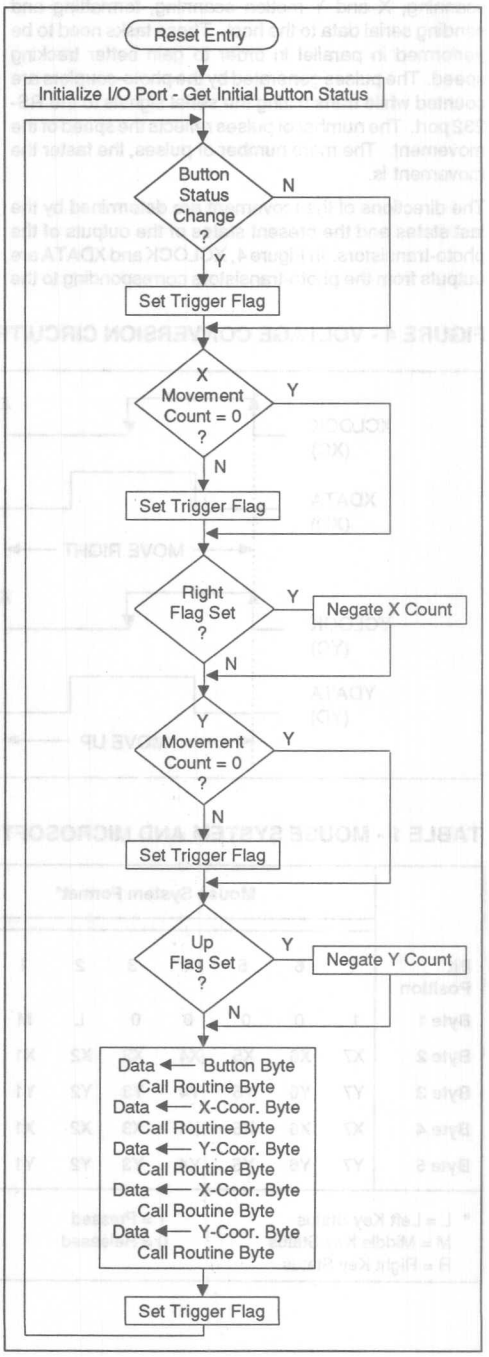
The mouse has been tested in Mouse System Mode and is functioning properly. A completed listing of the source program is given in Appendix A.

SUMMARY

The PIC16C54 from Microchip Technology Inc. provides a very cost-effective, high performance mouse implementation. Its low power (typically < 2 mA at 1 µs instruction cycle), small package (18-pin) and high reliability (on-chip watchdog timer to prevent software hangups) are among several reasons why the PIC16C54 is uniquely suitable for mouse applications.

This application note provides the user with a simple, fully functional serial mouse implementation. The user may use this as a starting point for a more comprehensive design. For fully implemented and compliant mouse products see Microchip's ASSP device family (MTA41XXX).

FIGURE 5 - FLOW CHART OF THE MAIN PROGRAM



© 1993 Microchip Technology Inc.

FIGURE 6 - FLOW CHART OF ROUTINE BYTE

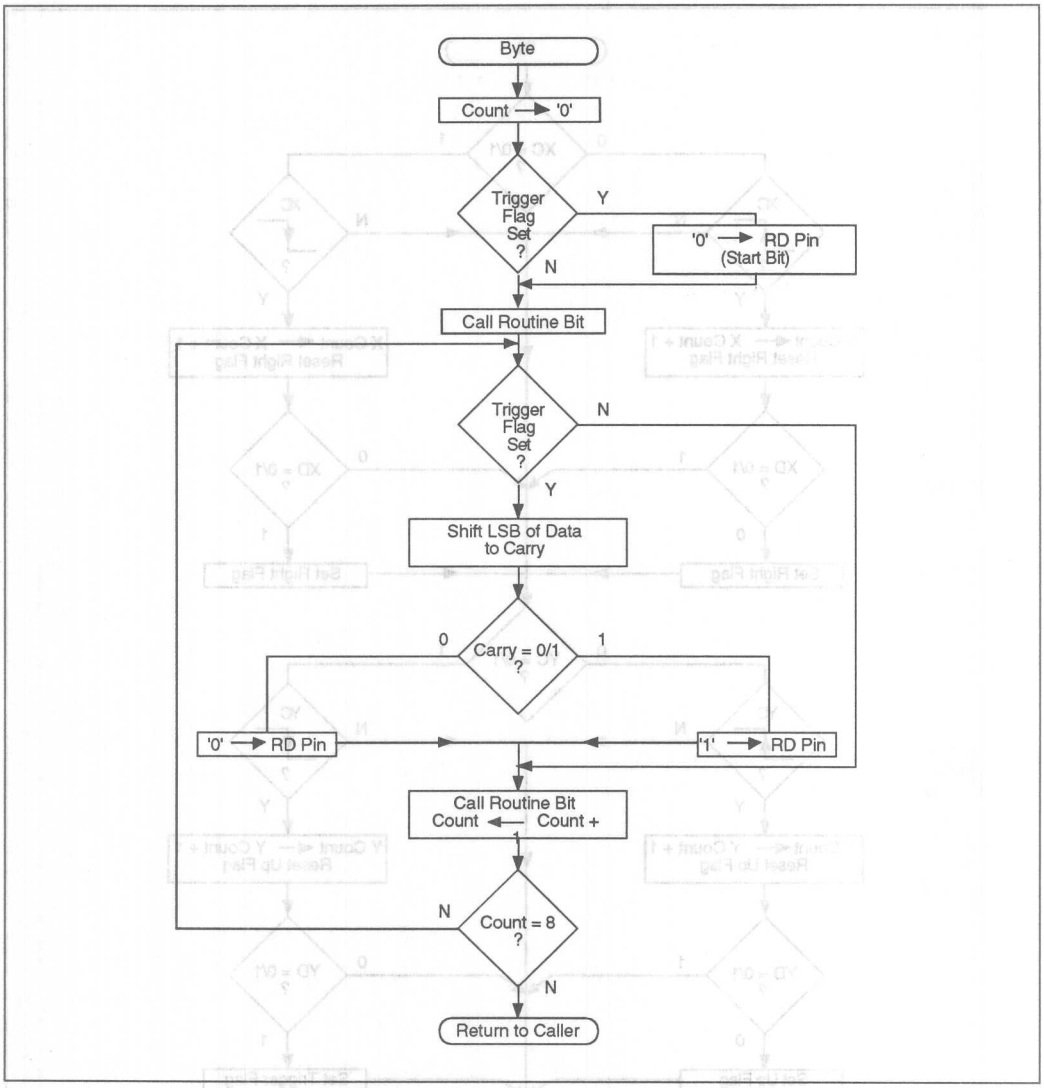
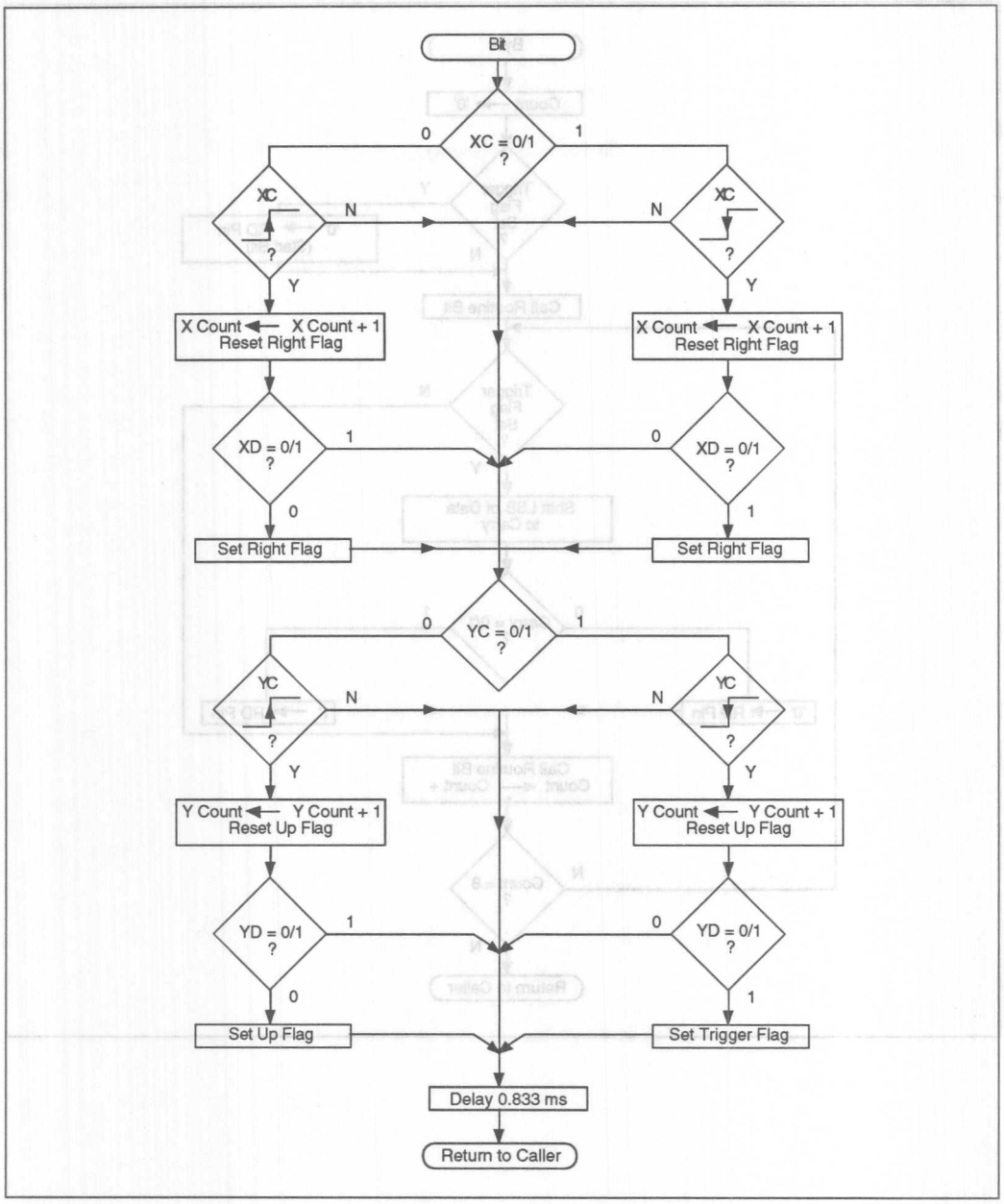


FIGURE 7 - FLOW CHART OF ROUTINE BIT



© 1993 Microchip Technology Inc.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1
 MOUSE

                      TITLE " MOUSE "
                       LIST   P=16C54,R=O
                      ;
                      ;*****************************************
                      ;*                                       *
                      ;*   MOUSE CONTROLLER                    *
                      ;*                                       *
                      ;*   VERSION : 25 APRIL, 1990            *
                      ;*                                       *
                      ;*   MODE = PIC16C54XT   CLK=4.0MHZ      *
                      ;*****************************************
                      ;
                      ;-------------------------
                      ;       FILES ASSIGNMENT
                      ;-------------------------
                      ;
0003                  STATUS        EQU 3           ;STATUS REGISTER
0005                  RA            EQU 5           ;I/O PORT A
0006                  RB            EQU 6           ;I/O PORT B
0008                  TIMER1        EQU 10          ;COUNTER FOR DELAY
000C                  CSTAT         EQU 14          ;CO-ORDINATE STATUS
000D                  BSTAT         EQU 15          ;BUTTON STATUS
000E                  DATA0         EQU 16          ;
000F                  DATA1         EQU 17          ;
0010                  DATA2         EQU 20          ;5 BYTE RS232 DATA
0011                  DATA3         EQU 21          ;
0012                  DATA4         EQU 22          ;
0013                  FLAGA         EQU 23          ;GENERAL PURPOSE FLAG
0014                  XCOUNT        EQU 24          ;X-MOVEMENT COUNTER
0015                  YCOUNT        EQU 25          ;Y-MOVEMENT COUNTER
0016                  FLAGB         EQU 26          ;GENERAL PURPOSE FLAG
0018                  COUNT         EQU 30          ;GENERAL PURPOSE COUNTER
0019                  DATA_AREA     EQU 31          ;FOR TEMP. STORAGE
                      ;
                      ;-------------------------
                      ;       BIT ASSIGNMENT
                      ;-------------------------
                      ;
0000                  YC            EQU 0           ;Y-CLOCK PIN
0001                  YD            EQU 1           ;Y-DATA PIN
0001                  UP            EQU 1           ;MOVING UP FLAG
0002                  XC            EQU 2           ;X-CLOCK PIN
0003                  XD            EQU 3           ;X-DATA PIN
0003                  RI            EQU 3           ;MOVING RIGHT FLAG
0000                  BU1           EQU 0           ;BUTTON #1 PIN
0002                  BU2           EQU 2           ;BUTTON #2 PIN
0000                  CA            EQU 0           ;CARRY FLAG
0007                  RD            EQU 7           ;RECEIVED DATA PIN TO RS232
0002                  ZERO_AREA     EQU 2           ;ZERO FLAG
0002                  TR            EQU 2           ;TIGGER FLAG
                      ;
                      ;=========================================
                      ;       SUBROUTINES
                      ;=========================================
                      ;
                      ;*****************************************
                       ORG 0
                      ;*****************************************
                      ;
                      ;=========================================
                      ; DELAY A BIT TIME AND CHECK XC & YC STATUS
                      ;=========================================
```

© 1993 Microchip Technology Inc.

```
                        BIT
0000 0745                   BTFSS   RA,XC           ;XC = 1 ?
0001 0A0A                   GOTO    BIT0
0002 064C                   BTFSC   CSTAT,XC        ;(XC=1)
0003 0A11                   GOTO    BITY            ;(XC ALWAYS = 1)
0004 02B4                   INCF    XCOUNT          ;(XC -|__)
0005 0476                   BCF     FLAGB,RI        ;DEFAULT LEFT
0006 0765                   BTFSS   RA,XD           ;LEFT / RIGHT ?
0007 0A11                   GOTO    BITY
0008 0576                   BSF     FLAGB,RI
0009 0A11                   GOTO    BITY
                        BIT0
000A 074C                   BTFSS   CSTAT,XC        ;(XC=0)
000B 0A11                   GOTO    BITY            ;(XC ALWAYS = 0)
000C 02B4                   INCF    XCOUNT          ;(XC __|-)
000D 0476                   BCF     FLAGB,RI        ;DEFAULT LEFT
000E 0665                   BTFSC   RA,XD           ;LEFT / RIGHT ?
000F 0A11                   GOTO    BITY
0010 0576                   BSF     FLAGB,RI
                        BITY
0011 0705                   BTFSS   RA,YC           ;YC = 1 ?
0012 0A1B                   GOTO    BITY0
0013 060C                   BTFSC   CSTAT,YC        ;(YC=1)
0014 0A22                   GOTO    BITDY           ;(YC ALWAYS = 1)
0015 02B5                   INCF    YCOUNT          ;(YC -|__)
0016 0436                   BCF     FLAGB,UP        ;DEFAULT DOWN
0017 0725                   BTFSS   RA,YD           ;DOWN / UP ?
0018 0A22                   GOTO    BITDY
0019 0536                   BSF     FLAGB,UP
001A 0A22                   GOTO    BITDY
                        BITY0
001B 070C                   BTFSS   CSTAT,YC        ;(YC=0)
001C 0A22                   GOTO    BITDY           ;(YC ALWAYS = 0)
001D 02B5                   INCF Y  COUNT           ;(YC __|-)
001E 0436                   BCF     FLAGB,UP        ;DEFAULT DOWN
001F 0625                   BTFSC   RA,YD           ;DOWN / UP ?
0020 0A22                   GOTO    BITDY
0021 0536                   BSF     FLAGB,UP
                        BITDY
0022 0205                   MOVF    RA,W            ;SAVE COOR. STATUS
0023 002C                   MOVWF   CSTAT
0024 0CC1                   MOVLW   193D            ;0.833 MS DELAY
0025 0028                   MOVWF   TIMER1
                        BITD0
0026 0000                   NOP
0027 02E8                   DECFSZ  TIMER1
0028 0A26                   GOTO    BITD0
0029 0800                   RETLW   0
                        ;
                        ;================================================
                        ;
                        ;************************************************
                        ;*      SUBROUTINE TO SEND A BYTE               *
                        ;*      AS RS232C FORMAT 8,N,1                  *
                        ;************************************************
                        ;
                        BYTE
002A 0078                   CLRF    COUNT           ;RESET 8 BIT COUNT
002B 0753                   BTFSS   FLAGA,TR        ;ANY TRIGGER
002C 0A2E                   GOTO    BYTE0
002D 04E6                   BCF     RB,RD           ;LOW RD FOR START BIT
                        BYTE0
002E 0900                   CALL    BIT
                        BYTE1
002F 0753                   BTFSS   FLAGA,TR        ;ANY TRIGGER ?
0030 0A37                   GOTO    BYTE3
0031 0339                   RRF     DATA_AREA       ;SHIFT DATA TO CARRY
0032 0703                   BTFSS   STATUS,CA       ;0 / 1 ?
0033 0A36                   GOTO    BYTE2
```


© 1993 Microchip Technology Inc.

```
0034 05E6                BSF     RB,RD               ;SEND A 1
0035 0A37                GOTO    BYTE3
                    BYTE2
0036 04E6                BCF     RB,RD               ;SEND A 0
                    BYTE3
0037 0900                CALL    BIT
0038 02B8                INCF    COUNT
0039 0778                BTFSS   COUNT,3             ;COUNT = 8 ?
003A 0A2F                GOTO    BYTE1
003B 0753                BTFSS   FLAGA,TR            ;ANY TRIGGER ?
003C 0A42                GOTO    BYTE4
003D 04E6                BCF     RB,RD               ;SEND SENT BIT
003E 0900                CALL    BIT
003F 05E6                BSF     RB,RD
0040 0900                CALL    BIT
0041 0A44                GOTO    BYTE5
                    BYTE4
0042 0900                CALL    BIT
0043 0900                CALL    BIT
                    BYTE5
0044 0800                RETLW   0
                    ;
                    ;===========================================
                    ;       RESET ENTRY
                    ;===========================================
                    ;
                    INIT
0045 0CC1                MOVLW   B'11000001'         ;DISABLE WATCH DOG
0046 0002                OPTION
0047 0C0F                MOVLW    B'00001111'        ;INIT RB0~3 BE INPUTS
0048 0006                TRIS RB                     ;RB4~7 BE OUTPUTS
0049 0CFF                MOVLW   B'11111111'         ;INIT RA0~3 BE INPUTS
004A 0005                TRIS    RA
004B 05E6                BSF     RB,RD               ;HIGH RD PIN
004C 0246                COMF    RB,W                ;GET INIT BUTTON INPUTS
004D 0E05                ANDLW   B'00000101'
004E 0D80                IORLW   B'10000000'
004F 002D                MOVWF   BSTAT
0050 002E                MOVWF   DATA0
0051 0205                MOVF    RA,W
0052 002C                MOVWF   CSTAT
0053 0073                CLRF    FLAGA               ;CLEAR TR FLAG
0054 0074                CLRF    XCOUNT              ;RESET XCOUNT & YCOUNT
0055 0075                CLRF    YCOUNT
                    SCAN
0056 006F                CLRF    DATA1               ;UPDATE X,Y MOVEMENT DATA
0057 0070                CLRF    DATA2
0058 0071                CLRF    DATA3
0059 0072                CLRF    DATA4
005A 0214                MOVF    XCOUNT,W            ;XCOUNT = 0 ?
005B 0743                BTFSS   STATUS,ZERO_AREA
005C 0A80                GOTO    WRITX
                    SCANA
005D 0215                MOVF Y  COUNT,W             ;YCOUNT = 0 ?
005E 0743                BTFSS   STATUS,ZERO_AREA
005F 0A92                GOTO    WRITY
                    SCANB
0060 0246                COMF    RB,W                ;BUTTON STATUS CHANGE ?
0061 0E05                ANDLW   B'00000101'
0062 0D80                IORLW   B'10000000'
0063 00AD                SUBWF   BSTAT
0064 0643                BTFSC   STATUS,ZERO_AREA    ;IF CHANGE THEN TRIGGER
0065 0A6B                GOTO    SCANC               ;(NO CHANGE)
0066 0553                BSF     FLAGA,TR            ;(CHANGE) SET TRIGGER FLAG
0067 0246                COMF    RB,W                ;FORMAT BUTTON STATUS DATA
0068 0E05                ANDLW   B'00000101'
0069 0D80                IORLW   B'10000000'
```

© 1993 Microchip Technology Inc.

```
006C 0E05                ANDLW   B'00000101'
006D 0D80                IORLW   B'10000000'
006E 002D                MOVWF   BSTAT
006F 020E                MOVF    DATA0,W          ;SEND DATA0,1,2,3,4 TO HOST
0070 0039                MOVWF   DATA_AREA
0071 092A                CALL    BYTE
0072 020F                MOVF    DATA1,W
0073 0039                MOVWF   DATA_AREA
0074 092A                CALL    BYTE
0075 0210                MOVF    DATA2,W
0076 0039                MOVWF   DATA_AREA
0077 092A                CALL    BYTE
0078 0211                MOVF    DATA3,W
0079 0039                MOVWF   DATA_AREA
007A 092A                CALL    BYTE
007B 0212                MOVF    DATA4,W
007C 0039                MOVWF   DATA_AREA
007D 092A                CALL    BYTE
007E 0453                BCF     FLAGA,TR         ;CLEAR TRIGGER FLAG
007F 0A56                GOTO    SCAN
                    ;
                    WRITX
0080 0553                BSF     FLAGA,TR         ;SET TRIGGER FLAG
0081 0C40                MOVLW   40H              ;IF XCOUNT > 64 THEN XCOUNT <-64
0082 0094                SUBWF   XCOUNT,W
0083 0603                BTFSC   STATUS,CA
0084 0A8D                GOTO    WRITR
                    WRITS
0085 0776                BTFSS   FLAGB,RI         ;LEFT / RIGHT ?
0086 0A90                GOTO    WRITL
0087 0274                COMF    XCOUNT           ;(RIGHT) NEG XCOUNT
0088 0294                INCF    XCOUNT,W
                    WRITA
0089 002F                MOVWF   DATA1
008A 0031                MOVWF   DATA3
008B 0074                CLRF    XCOUNT           ;RESET XCOUNT
008C 0A5D                GOTO    SCANA
                    ;
                    WRITR
008D 0C40                MOVLW   40H              ;XCOUNT <- 64
008E 0034                MOVWF   XCOUNT
008F 0A85                GOTO    WRITS
                    ;
                    WRITL
0090 0214                MOVF    XCOUNT,W         ;(LEFT)
0091 0A89                GOTO    WRITA
                    ;
                    WRITY
0092 0553                BSF     FLAGA,TR         ;SET TRIGGER FLAG
0093 0C40                MOVLW   40H              ;IF YCOUNT > 64 THEN YCOUNT <-64
0094 0095                SUBWF   YCOUNT,W
0095 0603                BTFSC   STATUS,CA
0096 0A9F                GOTO    WRITV
                    WRITW
0097 0736                BTFSS   FLAGB,UP         ;DOWN / UP ?
0098 0AA2                GOTO    WRITD
0099 0275                COMF    YCOUNT           ;(UP) NEG YCOUNT
009A 0295                INCF    YCOUNT,W
                    WRITB
009B 0030                MOVWF   DATA2
009C 0032                MOVWF   DATA4
009D 0075                CLRF    YCOUNT           ;RESET YCOUNT
009E 0A60                GOTO    SCANB
                    ;
```


© 1993 Microchip Technology Inc.

```
                        WRITV
009F 0C40                    MOVLW   40H                ;YCOUNT <- 64
00A0 0035                    MOVWF   YCOUNT
00A1 0A97                    GOTO    WRITW
                        ;
                        WRITD
00A2 0215                    MOVF    YCOUNT,W           ;(DOWN)
00A3 0A9B                    GOTO    WRITB
                        ;
                        ;=========================================
                        ;       RESET ENTRY
                        ;=========================================
                        ;
                         ORG 777
01FF 0A45                    GOTO INIT                  ;JUMP TO PROGRAM STARTING
                        ;
                        END
                        ;
                        ;*****************************************


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

AN531

Intelligent Remote Positioner (Motor Control)

Author: Steven Frank - Vesta Technology Inc.

INTRODUCTION

The excellent cost/performance ratio of the PIC16C5X are well suited for a low-cost proportional D.C. actuator controller. This application note depicts a design of a remote intelligent positioning system using a D.C. motor (up to 1/3 hp) run from 12 to 24 V. The position accuracy is one in eight bits or 0.4%. The PIC16C5X receives its command and control information via a MICROWIRE™ serial bus. However, any serial communication method is applicable.

IMPLEMENTATION

The PIC16C5X based controller receives commands from a host, compares them to the actual position, calculates the desired motor drive level and then pulses a full H-bridge (Figure 2). In this way it serves as a remote intelligent positioner, driving the load until it has reached the commanded position. It can be used to control any proportional D.C. actuator i.e. D.C. motor or proportional valve.

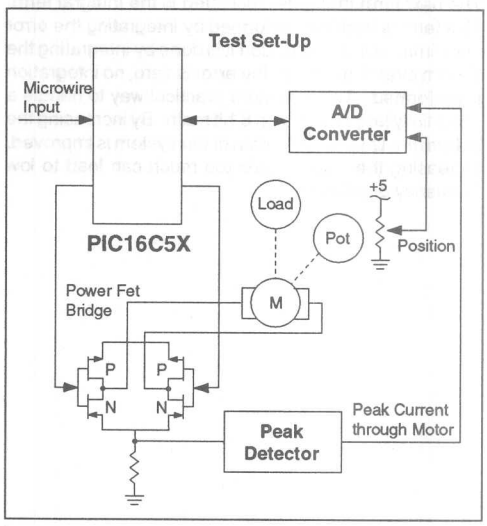
This system is ideally suited to remotely position valves and machinery. It can be used with D.C. motors to easily automate manual equipment. Because of the 5-wire serial interface, the positioner can be installed near its power supply and load. The remote intelligent positioner can then be linked to the central control processor by a small diameter easily routed cable. Since the positioner is running its own closed-loop PID algorithm (Figure 3), the host central processor need only send position commands and is therefore free to service the user interface, main application software and command many remote positioners.

The limit switch inputs provide a safety net this keeps the system from destroying itself in the event that the feedback device is lost. The optional current sense input can be used to determine if the load has jammed and prevent overheating of the actuator and drive electronics.

The commanded positions are presented to the PIC16C5X via a microwire type protocol at bit-rates of up to 50kb/s for the 4 MHz part. As currently implemented in this application note, the position request is the only communication. There are several variable locations available and they could be utilized to allow down-loading of the loop gain parameters, reading positioner information, or for setting a current limit. The host that is sending the position request must set the chip select low, and wait for the PIC16C5X to raise the "busy" (DO) line high. At this point, eight data bits can be clocked into the PIC16C5X. The requested position is sent most significant bit first and can be any 8-bit value. Values 1 through 255 represent valid positions with 0 being reserved for drive disable.

The PIC16C5X acquires data by way of a microwire A/D converter. This part was chosen for low cost yet it provides adequate performance. The second channel of the A/D is shown hooked up to a peak current detector. If the user desired, the PIC16C5X could monitor and protect the motor from overcurrent by monitoring this information.

FIGURE 1 - BLOCK DIAGRAM



MICROWIRE™ is a trademark of National Semiconductor Corporation.

© 1994 Microchip Technology Incorporated

The H-bridge power amplifier will deliver 10 or more amps at up to 24 volts when properly heat-sinked. It is wired for a modified 4-quadrant mode of operation. One leg of the bridge is used to control direction and the other leg pulses the low FET and the high FET alternately to generate the desired duty-cycle. In this way the system will operate well to produce a desired "speed" without the use of a separate speed control loop. This allows use of the PIC16C5X to control the PID algorithm for position directly while having reasonable speed control. The capacitance at the gates of the FETs combined with the impedance of the drive circuits provides for turn-off of the upper FET before the lower FET turns on... an important criteria.

The PID algorithm itself is where most of the meat of this application note is located so let's look at it more closely. The Algorithm is formed by summing the contribution of three basic components. The first calculation is the error for that is what the other terms are based on.

The error is the requested position minus the actual position. It is a signed number whose magnitude can be 255. In order not to lose resolution, the error is stored as an 8-bit magnitude with the sign stored separately in the FLAGS register under ER_SGN. This allows us to resolve a full signed 8-bit error with 8-bit math.

The proportional term is merely the algebraic difference of the requested position minus the actual position. It is scaled by a gain term (Kp) called the "proportional gain". The sign of this term is important for it tells the system which direction it must drive to correct the error. The proportional term is limited to ± 100. Increasing the proportional gain term will improve the dynamic and static accuracy of the system. Increasing it too much will cause oscillations.

The next term that gets calculated is the Integral term. This term is traditionally formed by integrating the error over time. In this application it is done by integrating the Ki term over time. When the error is zero, no integration is performed. This is a more practical way to handle a potentially large number in 8-bit math. By increasing the Ki term the D.C. or static gain of the system is improved. Increasing the integral gain too much can lead to low frequency oscillations.

The differential term (Kd) is a stabilizing term that helps keep the integral and proportional terms from overdriving the system through the desired position and thus creating oscillations. As you use more proportional and integral gain you will need more differential gain as well. The differential gain is calculated by looking at the rate of change of the positional error with respect to time. It is actually formed as "delta error/delta time" with the delta time being a program cycle.

The three terms are summed algebraically and scaled to produce a percentage speed request between 0 and 100%. The sign of the sum is used to control the H-bridge direction. The loop calculations run approximately 20 times per second on a 4 MHz part. This yields sufficient gain-bandwidth for most positioning applications. If higher system performance is desired, the number of pulses can be reduced to 20 and a 16 MHz PIC16C5X can be used. Your Loop gains (Kp, Ki, Kd) will have to be recalculated, but the system sample rate will be increased to 400 Hz. This should be sufficient to control a system that has a response time of 20 milliseconds or more.

The key to using the PIC16C5X series parts for PID control and PWM generation is to separate the two into separate tasks. There is simply not the hardware support or the processing speed to accurately do both concurrently. It is fortunate therefore that it is not necessary to do both concurrently. The systems that are generally controlled can be stabilized with a much lower information update rate than the PWM frequency. This supports the approach of calculating the desired percentage, outputting the PWM for a period of time and then recalculating the new desired percentage. Utilizing this technique the inexpensive PIC16C5X can implement PID control, PWM generation and still have processing time left over for monitor or communication functions.

About the Author:

Steven Frank has been designing analog and digital control systems for ten years. His background is in medical and consumer electronics. He has received numerous patents in control systems and instrumentation. At Vesta Technology Inc., Mr. Frank works with a number of engineers on custom embedded control systems designs. Vesta Technology Inc. is a provider of embedded control systems from an array of standard products and designs. Vesta offers custom design services and handles projects from concept to manufacturing.

© 1994 Microchip Technology Incorporated

FIGURE 2 - PROGRAM FLOW CHART

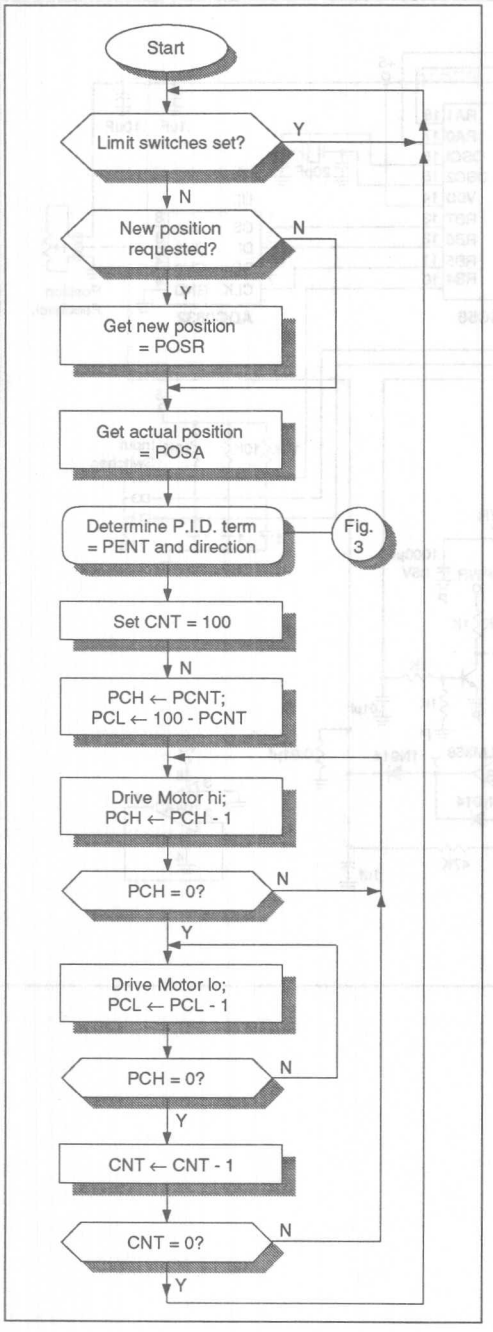
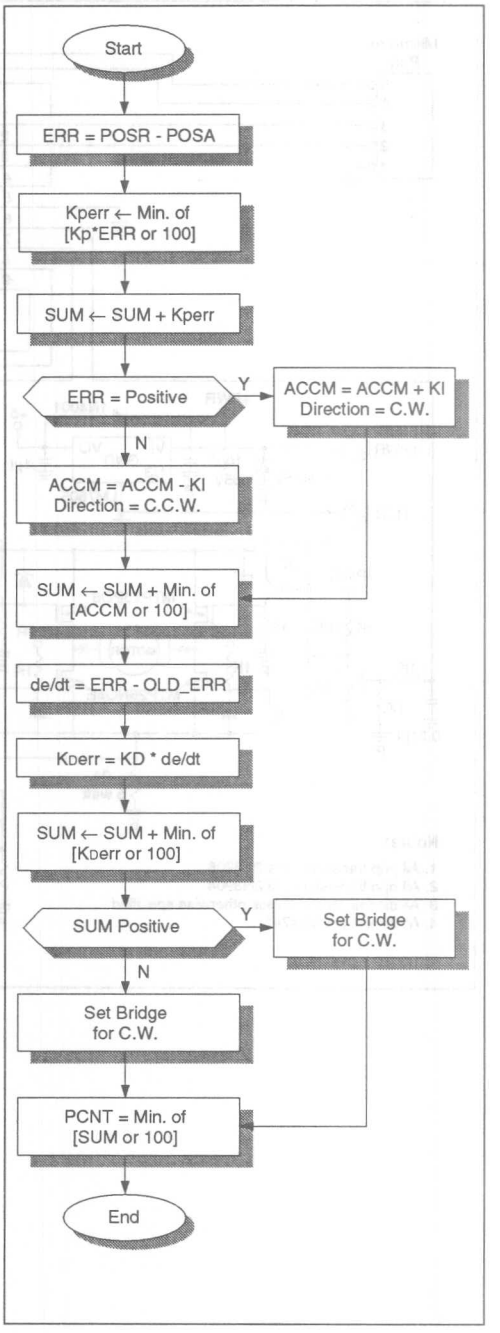


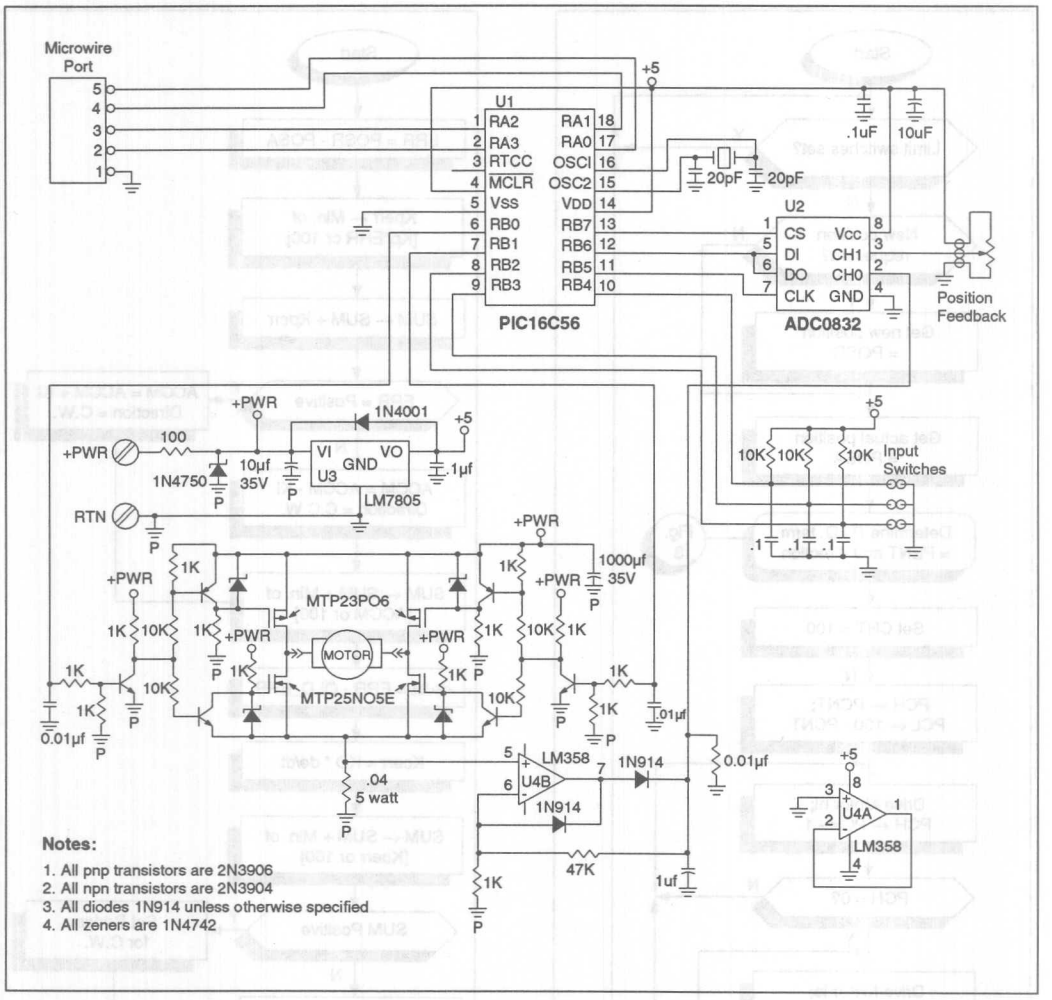
FIGURE 3 - P.I.D. ALGORITHM FLOW CHART



2

© 1994 Microchip Technology Incorporated

FIGURE 4 - SCHEMATIC



© 1994 Microchip Technology Incorporated

```
MPASM B0.54                                                    PAGE  1


                    ;
                    ;    mw8pos.asm
                    ;
                         LIST P=16C56
                    ;*****************************************************************
                    ; REV. A      Original release  1/10/92 srf
                    ;
                    ;*****************************************************************
                    ;
                    ;REGISTER EQUATES
                    ;
0000                 W       EQU     0
0000                 PNTR    EQU     00H             ; CONTENTS OF POINTER
0019                 FLAGS   EQU     19H             ; USE THIS VARIABLE LOCATION AS FLAGS
                                                     ; 0 BIT IS SIGN OF ERROR 1 IS NEGATIVE
                                                     ; 1 BIT IS SIGN OF ERROR ACCUMULATOR
                                                     ; 2 BIT IS SIGN OF THE DE/DE TERM
                                                     ; 3 BIT IS DIRECTION 0 IS CW
                                                     ; 4 BIT IS SIGN OF THE OLD ERROR
0003                 STATUS  EQU     03H
0003                 SWR     EQU     03H             ; STATUS WORD REGISTER
                                                     ; 0 = CARRY
                                                     ; 1 = DC
                                                     ; 2 = Z, SET IF RESULT IS ZERO
0004                 FSR     EQU     04H             ; FILE SELECT REGISTER
0005                 PORTA   EQU     05H             ; I/O REG (A0-A3), (A4-A7 DEF=0)
0006                 PORTB   EQU     06H             ; I/O REGISTER(B0-B7)
0007                 HI      EQU     07H             ; NUMBER OF HIGH MICROSECONDS
0008                 LO      EQU     08H             ; NUMBER OF LOW MICROSECONDS
0009                 PCNT    EQU     09H             ; PERCENT DUTYCYCLE REQUEST
000A                 HI_T    EQU     0AH             ; COUNTER FOR USECONDS LEFT/PULSE HI
000B                 LO_T    EQU     0BH             ; COUNTER FOR USECONDS LEFT/PULSE LO
000C                 ERROR   EQU     0CH             ; HOLDER FOR THE POSITIONAL ERROR
                                                     ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                                                     ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
000D                 SUMLO   EQU     0DH             ; PROGRESSIVE SUM OF THE PID TERMS
000E                 ACCUM   EQU     0EH             ; ERROR ACCUMULATOR
000F                 ERR_O   EQU     0FH             ; ERROR HISTORY USED FOR de/dt
                                                     ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                                                     ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
0010                 POSR    EQU     10H             ; POSITIONAL REQUEST
0011                 POSA    EQU     11H             ; ACTUAL POSITION
0012                 CYCLES  EQU     12H             ; COUNTER FOR CYCLES OUT

0013                 mulcnd  equ     13H             ; 8 bit multiplicand
0013                 ACCaLO  EQU     13H             ;same location used for the add routine
0014                 mulplr  equ     14H             ; 8 bit multiplier
0014                 ACCbLO  EQU     14H             ;same location used for the add routine
0015                 H_byte  equ     15H             ; High byte of the 16 bit result
0015                 ACCaHI  EQU     15H             ; same location used for the add routine
0016                 L_byte  equ     16H             ; Low byte of the 16 bit result
0016                 ACCbHI  EQU     16H             ; same location used for the add routine
0017                 count   equ     17H             ; loop counter
0018                 SUMHI   EQU     18H             ; HIGH BYTE OF THE LOOP SUM


                    ; PORT ASSIGNMENTS AND CONSTANTS

0000                 PWMCW   EQU     0               ; CLOCKWISE PWM OUTPUT BIT
0001                 PWMCCW  EQU     1               ; COUNTERCLOCKWISE PWM OUTPUT BIT
0000                 CARRY   EQU     0               ; CARRY BIT IN THE STATUS REGISTER
0002                 Z       EQU     2               ; THE ZERO BIT OF THE STATUS REGISTER
0001                 Same    equ     1               ;
```

© 1994 Microchip Technology Incorporated

```
0000                ER_SGN  EQU     0               ; SIGN BIT FOR THE ERROR IN FLAG REGISTER
0001                AC_SGN  EQU     1               ; SIGN BIT FOR THE ERROR ACCUMULATOR
0002                DE_SGN  EQU     2               ; SIGN BIT FOR DE/DT
0004                OER_SGN EQU     4               ; SIGN BIT FOR THE OLD ERROR
0030                KP      EQU     30              ; PROPORTIONAL GAIN
0002                KI      EQU     2               ; INTEGRAL GAIN
0020                KD      EQU     20              ; DIFFERENTIAL GAIN
0003                DIR     EQU     3               ; THE DIRECTION FLAG
0007                CSN     EQU     7               ; CHIP SELECT NOT ON A/D
0006                BV      EQU     6               ; DATA LINE FOR THE A/D
0005                CK      EQU     5               ; CLOCK LINE FOR THE A/D
0002                MWDO    EQU     2               ; MICROWIRE DATA OUT FROM POSITIONER
0001                MWDI    EQU     1               ; MICROWIRE DATA IN TO POSITIONER
0000                MWCS    EQU     0               ; MICROWIRE CHIP SELECT TO POSITIONER
0003                MWCK    EQU     3               ; MICROWIRE CLOCK IN TO POSITIONER

                    ;***** MACROS ************************************************
                    ;
                    CLKUP   MACRO                   ; clock up macro for the microwire
                      BSF     PORTB,CK              ; data acquisition from the a/d
                      NOP
                      ENDM

                    CLKDN   MACRO                   ; clock down macro for the microwire
                      BCF     PORTB,CK              ; data acquisition from the a/d
                      NOP
                      ENDM

                    GET_BIT MACRO                   ; ** FOR RECEIVING A/D DATA **
                      BCF     SWR,CARRY
                      BSF     PORTB,CK              ; SET CLOCK BIT HIGH
                      BTFSC   PORTB,BV              ; LOOK AT DATA COMMING IN
                      BSF     SWR,CARRY             ; SET THE CARRY FOR A 1
                      RLF     POSA                  ; ROTATE THE W REG LEFT
                      BCF     PORTB,CK              ; SET THE CLOCK LOW
                      NOP                           ; DELAY
                      ENDM

0000 0B88           GOTO    CLRREG

                    ;***** MATH ROUTINES *****************************************
                    ;
                    ; **** 8 BIT MULTIPLY ********
                    ; ****************************        Begin Multiplier Routine
0001 0075           mpy_S   clrf    H_byte
0002 0076                   clrf    L_byte
0003 0C08                   movlw   8
0004 0037                   movwf   count
0005 0213                   movf    mulcnd,w
0006 0403                   bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
0007 0334           loop    rrf     mulplr
0008 0603                   btfsc   STATUS,CARRY
0009 01F5                   addwf   H_byte,Same
000A 0335                   rrf     H_byte,Same
000B 0336                   rrf     L_byte,Same
000C 02F7                   decfsz  count
000D 0A07                   goto    loop
000E 0800                   retlw   0

                    ;****************************
                    ; DOUBLE PRECISION ADD AND SUBTRACT ( ACCb-ACCa->ACCb )

000F 0917           D_sub   call    neg_A           ; At first negate ACCa, then add

                    ;***************
                    ; Double Precision Addition ( ACCb+ACCa->ACCb )

0010 0213           D_add   movf    ACCaLO,W
0011 01F4                   addwf   ACCbLO          ; add lsb
```


© 1994 Microchip Technology Incorporated

```
0012 0603                btfsc   STATUS,CARRY    ; add in carry
0013 02B6                incf    ACCbHI
0014 0215                movf    ACCaHI,W
0015 01F6                addwf   ACCbHI          ; add msb
0016 0800                retlw   00
                ;
                ;
0017 0273       neg_A    comf    ACCaLO          ; negate ACCa
0018 02B3                incf    ACCaLO
0019 0643                btfsc   STATUS,Z
001A 00F5                decf    ACCaHI
001B 0275                comf    ACCaHI
001C 0800                retlw   00
                ; ****************************************
                ; divide by 16 and limit to 100 Decimal

                SHIFT   MACRO
                  BCF     SWR,CARRY
                  RRF     L_byte
                  BCF     SWR,CARRY
                  RRF     H_byte
                  BTFSC   SWR,CARRY
                  BSF     L_byte,7
                  ENDM
                DIV_LMT
                  SHIFT
001D 0403                BCF     SWR,CARRY
001E 0336                RRF     L_byte
001F 0403                BCF     SWR,CARRY
0020 0335                RRF     H_byte
0021 0603                BTFSC   SWR,CARRY
0022 05F6                BSF     L_byte,7
                  SHIFT
0023 0403                BCF     SWR,CARRY
0024 0336                RRF     L_byte
0025 0403                BCF     SWR,CARRY
0026 0335                RRF     H_byte
0027 0603                BTFSC   SWR,CARRY
0028 05F6                BSF     L_byte,7
                  SHIFT
0029 0403                BCF     SWR,CARRY
002A 0336                RRF     L_byte
002B 0403                BCF     SWR,CARRY
002C 0335                RRF     H_byte
002D 0603                BTFSC   SWR,CARRY
002E 05F6                BSF     L_byte,7
                  SHIFT
002F 0403                BCF     SWR,CARRY
0030 0336                RRF     L_byte
0031 0403                BCF     SWR,CARRY
0032 0335                RRF     H_byte
0033 0603                BTFSC   SWR,CARRY
0034 05F6                BSF     L_byte,7

                LMT100
0035 0C01                MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0036 0095                SUBWF   H_byte,0        ; IF THERE IS ANYTHING THERE, IF NOT,
0037 0703                BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0038 0A3C                GOTO    L8_E            ; OTHERWISE GIVE THE LOW BYTE A FULL
0039 0C64                MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
003A 0036                MOVWF   L_byte          ; TO 100
003B 0A42                GOTO    LMT_EXIT
                L8_E
003C 0C64                MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
```

© 1994 Microchip Technology Incorporated

```
003D 0096                SUBWF   L_byte,0        ; 100 DECIMAL
003E 0703                BTFSS   SWR,CARRY
003F 0A42                GOTO    LMT_EXIT
0040 0C64                MOVLW   64H
0041 0036                MOVWF   L_byte
                    LMT_EXIT
0042 0800                RETLW   00
                    ;
                    ;THE ROUTINE CALCTIMES DOES THE FOLLOWING: PCNT = DUTY CYCLE IN %
                    ; 100 - PCNT -> LO  AND PCNT -> HI. ZERO VALUES IN EITHER LO OR HI
                    ;ARE FORCED TO 1.
                    CALCTIMES
0043 0209                MOVF    PCNT,W          ; PUT REQUESTED % INTO W REGISTER
0044 0027                MOVWF   HI              ; COPY ON MICROSECONDS IN TO HI TIME
0045 0C64                MOVLW   64H
0046 0028                MOVWF   LO
0047 0209                MOVF    PCNT,0
0048 00A8                SUBWF   LO,1            ; LEAVE 100-HI TIME IN LO TIME
0049 0207                MOVF    HI,0            ; INSPECT THE HIGH TIME
004A 0643                BTFSC   SWR,2           ; IF ITS IS ZERO
004B 02A7                INCF    HI,1            ; INCREMENT IT
004C 0208                MOVF    LO,0            ; INSPECT THE LO TIME
004D 0643                BTFSC   SWR,2           ; IF ITS ZERO
004E 02A8                INCF    LO,1            ; INCREMENT IT
004F 0800                RETLW   00


                    ;*****************************************************************
                    BEGIN
0050 0000                NOP                     ; STUBBED BEGINNING


                    ;****CHECKING THE LIMIT SWITCHES AND CHECKING FOR MW***************
                    ; This will check the switch inputs for closure and will terminate
                    ; pulsing is one is closed.  It doesn't distinguish between the switches
                    ; so they are not dedicated to cw end and ccw end.

                    SW_TRAP
0051 0004                CLRWDT
0052 0746                BTFSS   PORTB,2         ; THIS WILL TEST ALL THREE OF THE
0053 0A51                GOTO    SW_TRAP         ; SWITCH INPUTS.  IF ANY ONE IS
0054 0766                BTFSS   PORTB,3         ; SET THEN EXECUTION OF THE CODE
0055 0A51                GOTO    SW_TRAP         ; WILL BE LIMITED TO LOOKING FOR
0056 0786                BTFSS   PORTB,4         ; IT TO BE CLEARED
0057 0A51                GOTO    SW_TRAP


                    ;****RECEIVING THE POSITIONAL REQUEST*****************************
                    ; The host system that wishes to send positional requests to the positioner
                    ; servo makes its desire known by setting the chip select to the positioner
                    ; low.  It then monitors the busy (Data Out) line from the positioner.  When
                    ; the positioner sets the busy line high, the host may begin sending its 8
                    ; request.  The data bits should be valid on the rising edge of the clock.
                    ; After 8 bits have been received by the positioner it will begin operation
                    ; to send the system to the received position.  It can be interrupted at any
                    ; point during the positioning process by the host sending a new command.
                    ; opportunity to update the command is issued every 100 pwm pulses (every 50
                    ; milliseconds).
                    ; If the host sends a zero positional command the positioner will stop the
                    ; system and remain inactive.
                    ; If the host does not successfully complete a microwire transmission of 8
                    ; data bits the watchdog timer will trip and reset the system to an inactive
                    ; "stopped" state.
                    REC_MW
0058 0C0B                MOVLW   0BH             ; RESET THE PORT FOR THREE INPUTS
```


© 1994 Microchip Technology Incorporated

```
0059 0005                TRIS    PORTA           ; AND ONE OUTPUT
005A 0445                BCF     PORTA,MWDO      ; SET THE DATA OUT LOW FOR BUSY
005B 0C20                MOVLW   20H
005C 0037                MOVWF   count
                    WATCH_CS
005D 0705                BTFSS   PORTA,MWCS      ; CHECK FOR INCOMMING REQUESTS
005E 0A62                GOTO    REC_CMD         ; RECEIVE A NEW POSITION REQUEST
005F 02F7                DECFSZ  count,1
0060 0A5D                GOTO    WATCH_CS
0061 0A71                GOTO    REC_EXIT        ; NO REQUEST WAS MADE IN THE TIME ALLOTED
                    REC_CMD
0062 0545                BSF     PORTA,MWDO      ; SET THE DATA OUT HIGH FOR "OK TO SEND"
0063 0C08                MOVLW   8H              ; SET TO RECEIVE 8 BITS
0064 0037                MOVWF   count
                    WAIT_UP
0065 0765                BTFSS   PORTA,MWCK      ; WAIT FOR A RISING EDGE
0066 0A65                GOTO    WAIT_UP
0067 0403                BCF     SWR,CARRY       ; RESET THE CARRY TO A DEFAULT ZERO
0068 0625                BTFSC   PORTA,MWDI      ; READ THE DATA IN
0069 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A ONE
006A 0370                RLF     POSR,1          ; ROTATE THE BIT INTO THE POSITION REQ.
006B 02F7                DECFSZ  count,1         ; DECREMENT THE BIT COUNTER
006C 0A6E                GOTO    WAIT_DN         ; WAIT FOR THE FALLING EDGE
006D 0A71                GOTO    REC_EXIT        ; LAST BIT RECEIVED
                    WAIT_DN
006E 0665                BTFSC   PORTA,MWCK      ; CHECK THE INCOMMING CLOCK
006F 0A6E                GOTO    WAIT_DN         ; IF IT IS STILL HIGH WAIT FOR IT TO GO LOW
0070 0A65                GOTO    WAIT_UP         ; IF IT GOES LOW GO BACK TO RECEIVE NEXT BIT

                    REC_EXIT
0071 0445                BCF     PORTA,MWDO      ; SET THE BUSY FLAG

                    ;********** CHECK FOR THE DISABLE REQUEST ************************
                    ; Position 0 is considered a request to not drive the system.  In this way
                    ; the positioner will come up from a reset in a safe state and will not
                    ; try to move the system to some arbitrary location.

                    MOVE?
0072 0210                MOVF    POSR,W          ; CHECK THE REQUESTED POSTION
0073 0643                BTFSC   SWR,Z           ; IF IT IS ZERO THEN WAIT FOR A NON-ZERO
0074 0A50                GOTO    BEGIN           ; REQUEST BY BRANCHING BACK TO THE BEGINNING

                    ;****READING THE A/D VALUES*****************************************
                    ;
                    ; Read the positional a/d channel (1) and store the value in the actual
                    ; position variable (POSA).
                    ; This is written in line to minimize the use of variables

                    READ_POS
0075 0071                CLRF    POSA            ; CLEAN THE POSITION ACTUAL HOLDER
0076 04E6                BCF     PORTB,CSN       ; SET THE CHIP SELECT LOW TO A/D
0077 0C1C                MOVLW   1CH             ; SET THE DATA LINE TO OUTPUT
0078 0006                TRIS    PORTB           ; FOR SENDING SET-UP BITS
0079 05C6                BSF     PORTB,BV        ; SET FOR "START" BIT
007A 0000                NOP
                      CLKUP                      ; CLOCK IN THE START BIT
007B 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
007C 0000                NOP

                      CLKDN                      ; "
007D 04A6                BCF     PORTB,CK        ; data acquisition from the a/d
007E 0000                NOP

                      CLKUP                      ; CLOCK IN SINGLE-ENDED
007F 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
0080 0000                NOP

                      CLKDN                      ; "
```

```
0081 04A6                BCF     PORTB,CK        ; data acquisition from the a/d
0082 0000                NOP

                    CLKUP                        ; CLOCK IN CHANNEL 1
0083 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
0084 0000                NOP
                    CLKDN                        ; TO THE MUX
0085 04A6                BCF     PORTB,CK        ; data acquisition from the a/d
0086 0000                NOP

0087 0C5C                MOVLW   5CH             ; SET THE DATA LINE TO INPUT
0088 0006                TRIS    PORTB           ; TO RECEIVE DATA BITS FROM A/D
                    CLKUP                        ; CLOCK UP TO LET MUX SETTLE
0089 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
008A 0000                NOP

                    CLKDN                        ; CLOCK DN TO LET MUX SETTLE
008B 04A6                BCF     PORTB,CK        ; data acquisition from the a/d
008C 0000                NOP

                    GET_BIT                      ; GET BIT 7
008D 0403                BCF     SWR,CARRY
008E 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
008F 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0090 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0091 0371                RLF     POSA            ; ROTATE THE W REG LEFT
0092 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
0093 0000                NOP                     ; DELAY

                    GET_BIT                      ; BIT 6
0094 0403                BCF     SWR,CARRY
0095 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
0096 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0097 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0098 0371                RLF     POSA            ; ROTATE THE W REG LEFT
0099 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
009A 0000                NOP                     ; DELAY
                    GET_BIT                      ; BIT 5
009B 0403                BCF     SWR,CARRY
009C 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
009D 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
009E 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
009F 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00A0 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00A1 0000                NOP                     ; DELAY

                    GET_BIT                      ; BIT 4
00A2 0403                BCF     SWR,CARRY
00A3 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00A4 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00A5 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00A6 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00A7 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00A8 0000                NOP                     ; DELAY

                    GET_BIT                      ; BIT 3
00A9 0403                BCF     SWR,CARRY
00AA 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00AB 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00AC 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00AD 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00AE 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00AF 0000                NOP                     ; DELAY

                    GET_BIT                      ; BIT 2
00B0 0403                BCF     SWR,CARRY
00B1 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00B2 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
```


© 1994 Microchip Technology Incorporated

```
00B3 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00B4 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00B5 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00B6 0000                NOP                     ; DELAY
                    GET_BIT                      ; BIT 1
00B7 0403                BCF     SWR,CARRY
00B8 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00B9 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00BA 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00BB 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00BC 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00BD 0000                NOP                     ; DELAY

                    GET_BIT                      ; BIT 0
00BE 0403                BCF     SWR,CARRY
00BF 05A6                BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00C0 06C6                BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00C1 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00C2 0371                RLF     POSA            ; ROTATE THE W REG LEFT
00C3 04A6                BCF     PORTB,CK        ; SET THE CLOCK LOW
00C4 0000                NOP                     ; DELAY

00C5 05E6                BSF     PORTB,CSN       ; DESELECT THE CHIP


                    ;***************** CALCULATING THE PID TERMS **********************
                    ;****CALCULATE THE ERROR*******
                    ; The error is very simply the signed difference between where the
                    ; system is and where it is supposed to be at a particular instant
                    ; in time.  It is formed by subtracting the actual position from the
                    ; requested position (Position requested - Position actual).  This
                    ; difference is then used to determine the proportional,integral and
                    ; differential term contributions to the output.
                    C_ERR
00C6 0211                MOVF    POSA,0          ; LOAD THE ACTUAL POSITION INTO W
00C7 0090                SUBWF   POSR,0          ; SUBTRACT IT FROM THE REQUESTED POSITION
00C8 0603                BTFSC   SWR,CARRY       ; CHECK THE CARRY BIT TO DETERMINE THE SIGN
00C9 0ACB                GOTO    PLS_ER          ; ITS POSATIVE  (POSR>POSA)
00CA 0ACE                GOTO    MNS_ER          ; ITS NEGATIVE  (POSA>POSR)

                    PLS_ER
00CB 002C                MOVWF   ERROR           ; SAVE THE DIFFERENCE IN "ERROR"
00CC 0419                BCF     FLAGS,ER_SGN    ; SET THE SIGN FLAG TO INDICATE POSATIVE
00CD 0AD2                GOTO    CE_EXIT

                    MNS_ER
00CE 0210                MOVF    POSR,0          ; RE-DO THE SUBTRACTION
00CF 0091                SUBWF   POSA,0          ; ACTUAL - REQUESTED
00D0 002C                MOVWF   ERROR           ; STORE THE DIFFERENCE IN "ERROR"
00D1 0519                BSF     FLAGS,ER_SGN    ; SET THE SIGN FLAG FOR NEGATIVE

                    CE_EXIT
00D2 006D                CLRF    SUMLO           ; CLEAN OLD VALUES OUT TO PREPARE
00D3 0078                CLRF    SUMHI           ; FOR THIS CYCLES SUMMATION

                    ;****CALCULATE THE PROPORTIONAL TERM******
                    ; The proportional term is the error times the proportional gain term.
                    ; This term simply gives you more output drive the farther away you are
                    ; from where you want to be (error)*Kp.
                    ; The proportional gain term is a signed term between -100 and 100  The
                    ; more proportional gain you have the lower your system following error
                    ; will be.  The higher your proportional gain, the more integral and
                    ; differential term gains you will have to add to make the system stable.
                    ; The sum is being carried as a 16 bit signed value.
                    C_PROP
```

© 1994 Microchip Technology Incorporated

```
00D4 020C                MOVF    ERROR,0         ; LOAD THE ERROR TERM INTO W
00D5 0033                MOVWF   mulcnd          ; MULTIPLY IT BY THE PROPORTIONAL GAIN
00D6 0C30                MOVLW   KP              ; KP AND THEN SCALE IT DOWN BY DIVIVING
00D7 0034                MOVWF   mulplr          ; IT DOWN BY 16. IF IT IS STILL OVER
00D8 0901                CALL    mpy_S           ; 255 THEN LIMIT IT TO 255
00D9 091D                CALL    DIV_LMT

                    RESTORE_SGN
00DA 0719                BTFSS   FLAGS,ER_SGN    ; IF THE ERROR SIGN IS NEGATIVE THEN
00DB 0ADE                GOTO    ADDPROP         ; PUT THE SIGN INTO THE LOW BYTE
00DC 0276                COMF    L_byte,1
00DD 02B6                INCF    L_byte,1

                    ADDPROP
00DE 0216                MOVF    L_byte,W        ; SAVE THE PROPORTIONAL PART
00DF 01ED                ADDWF   SUMLO,1         ; IN THE SUM
00E0 0603                BTFSC   SWR,CARRY       ; IF THE ADDITION CARRIED OUT THEN
00E1 02B8                INCF    SUMHI,1         ; INCREMENT THE HIGH BYTE
00E2 0C00                MOVLW   0               ; THEN
00E3 06ED                BTFSC   SUMLO,7         ; SIGN EXTEND TO THE UPPER
00E4 0CFF                MOVLW   0FF             ; BYTE
00E5 01F8                ADDWF   SUMHI,1


                    ;****CALCULATE THE INTEGRAL TERM******
                    ; The integral term is an accumulation of the error thus far.  Its purpose
                    ; is to allow even a small error to effect a large change.  It does this
                    ; by adding a small number into an accumulator each cycle through the pro
                    ; Thusly even a small error that exist for a while will build up to a large
                    ; enough number to effect an output sufficient to move the system.  The
                    ; that this integral accumulator has is modulated by the integral gain term
                    ; The integral of the error over time is multiplied be KI and the result is
                    ; contribution to the final summation for determining the output value.  This
                    ; term helps to insure the long-term accuracy of the system is good.  A
                    ; amount is necessary for this purpose but too much will cause oscillations.
                    ; The integral is bounded in magnitude for two purposes.  The first is so
                    ; it never rolls over and changes sign.  The second is that it may saturate
                    ; long moves forcing an excessively large overshoot to "de-integrate" the
                    ; accumulated during the first of the move

                    C_INT
00E6 020C                MOVF    ERROR,W         ; MOVE THE ERROR INTO THE W REG
00E7 0643                BTFSC   SWR,Z           ; AND CHECK TO SEE IF IT IS ZERO
00E8 0AFF                GOTO    ADDINT          ; IF SO THEN DONT CHANGE THE ACCUMULATOR
00E9 0619                BTFSC   FLAGS,ER_SGN    ; TEST THE FLAGS TO FIND THE POLARITY
00EA 0AEE                GOTO    MNS_1           ; OF THE ERROR .. 0 POSATIVE 1 NEGATIVE
                    PLS_1
00EB 0C02                MOVLW   KI              ; IF POSATIVE ADD ONE TO
00EC 01EE                ADDWF   ACCUM,1         ; THE ERROR ACCUMULATOR
00ED 0AF0                GOTO    LMTACM          ; THEN LIMIT IT TO +/-100
                    MNS_1
00EE 0C02                MOVLW   KI              ; IF NEGATIVE THEN SUBTRACT ONE
00EF 00AE                SUBWF   ACCUM,1         ; FROM THE ERROR ACCUMULATOR
                    LMTACM
00F0 06EE                BTFSC   ACCUM,7         ; CHECK THE SIGN BIT OF THE ERROR ACCUMULA-
TOR
00F1 0AF9                GOTO    M_LMT           ; AND DO A POSATIVE OR NEGATIVE LIMIT
                    P_LMT
00F2 0C9C                MOVLW   9CH             ; FOR THE POSATIVE LIMIT ADD 156 TO THE
00F3 01CE                ADDWF   ACCUM,0         ; NUMBER AND SEE IF YOU GENERATE A CARRY
00F4 0703                BTFSS   SWR,CARRY       ; BY CHECKING THE CARRY FLAG
00F5 0AFF                GOTO    ADDINT          ; IF NOT THEN ITS O.K.
00F6 0C64                MOVLW   64H             ; IF SO THEN FORCE THE ACCUMULATOR TO
00F7 002E                MOVWF   ACCUM           ; 100 DECIMAL
00F8 0AFF                GOTO    ADDINT
                    M_LMT
00F9 0C9C                MOVLW   9CH             ; FOR THE NEGATIVE LIMIT SUBTRACT 156 FROM
00FA 008E                SUBWF   ACCUM,0         ; THE NUMBER AND SEE IF YOU GENERATE A
```


© 1994 Microchip Technology Incorporated

```
00FB 0603                BTFSC   SWR,CARRY       ; NON-CARRY CONDITION INDICATING A ROLL-OVER
00FC 0AFF                GOTO    ADDINT          ; IF NOT THEN LEAVE THE ACCUMULATOR ALONE
00FD 0C9C                MOVLW   9CH             ; IF SO THEN LIMIT IT TO -100 BY
00FE 002E                MOVWF   ACCUM           ; FORCING THAT VALUE IN THE ACCUMULATOR

                    ADDINT
00FF 020E                MOVF    ACCUM,W         ; ADD THE INTEGRAL ACCUMULATOR TO
0100 01ED                ADDWF   SUMLO,1         ; THE LOW BYTE OF THE SUM
0101 0603                BTFSC   SWR,CARRY       ; TEST FOR OVERFLOW, IF SO THEN
0102 02B8                INCF    SUMHI,1         ; INCREMENT THE HI BYTE
0103 0C00                MOVLW   0               ; LOAD 0 INTO THE W REGISTER
0104 06EE                BTFSC   ACCUM,7         ; IF THE INTEGRAL ACCUMULATOR WAS NEGATIVE
0105 0240                COMF    W,W             ; COMPLEMENT THE 0 TO GET SIGN FOR HIGH BYTE
0106 01F8                ADDWF   SUMHI,1         ; ADD INTO THE HIGH BYTE OF THE SUM

                    U_DEXIT                      ; EXIT POINT FOR THE UP/DOWN CONTROL OF ACCUM


                    ;****CALCULATING THE DIFFERENTIAL TERM*************************
                    ; The differential term examines the error and determines how much
                    ; it has changed since the last cycle.  It does this by subtracting the
                    ; old error from the new error.  Since the cycle time is relatively fixed
                    ; we can use it as the "dt" of the desired "de/dt".  This derivative of the
                    ; error is then multiplied by the differential gain term KD and becomes the
                    ; differential term contribution for the final summation.

                    ; First, create the "de" term by doing a signed subtaction of new error
                    ; minus the old error. (new_error - old_error)
                    C_DIFF
0107 020C                MOVF    ERROR,W         ; LOAD THE NEW ERROR INTO REGISTER
0108 0719                BTFSS   FLAGS,ER_SGN
0109 0B0D                GOTO    LO_BYTE
010A 026C                COMF    ERROR,1         ; CORRECT THE VALUE TO BE 16 BIT
010B 028C                INCF    ERROR,W
010C 026C                COMF    ERROR,1         ; RESTORE IT FOR FUTURE USE TO 8 BIT MAGNI-
TUDE
                    LO_BYTE
010D 0034                MOVWF   ACCbLO          ; FOR SUBTRACTION
010E 0C00                MOVLW   00
010F 0619                BTFSC   FLAGS,ER_SGN    ; SIGN EXTEND THE UPPER BYTE
0110 0CFF                MOVLW   0FF
0111 0036                MOVWF   ACCbHI
0112 020F                MOVF    ERR_O,W         ; LOAD THE OLD ERROR INTO OTHER REGISTER
0113 0799                BTFSS   FLAGS,OER_SGN
0114 0B17                GOTO    LO_BYTEO
0115 026F                COMF    ERR_O,1         ; CORRECT THE VALUE TO BE 16 BIT
0116 028F                INCF    ERR_O,W
                    LO_BYTEO
0117 0033                MOVWF   ACCaLO          ; FOR SUBTRACTION
0118 0C00                MOVLW   00
0119 0699                BTFSC   FLAGS,OER_SGN   ; SIGN EXTEND THE UPPER BYTE
011A 0CFF                MOVLW   0FF
011B 0035                MOVWF   ACCaHI
011C 090F                CALL    D_sub           ; PERFORM THE SUBTRACTION

                    STRIP_SGN
011D 06F6                BTFSC   ACCbHI,7        ; TEST THE SIGN OF THE RESULT
011E 0B20                GOTO    NEG_ABS
011F 0B25                GOTO    POS_ABS
                    NEG_ABS
0120 0559                BSF     FLAGS,DE_SGN    ; ITS NEGATIVE SO SET THE FLAG AND
0121 0274                COMF    ACCbLO,1        ; COMPLEMENT THE VALUE
0122 0294                INCF    ACCbLO,W
0123 002F                MOVWF   ERR_O
0124 0B28                GOTO    MULT_KD
                    POS_ABS
0125 0459                BCF     FLAGS,DE_SGN    ; ITS POSATIVE SO SET RESET THE FLAG
```

© 1994 Microchip Technology Incorporated

```
                ; Then multiply by Kd

                MULT_KD
0128 020F               MOVF    ERR_O,W
0129 0033               MOVWF   mulcnd          ; MOVE THE DE/DT TERM INTO THE MULCND REG.
012A 0C20               MOVLW   KD              ; MOVE THE DIFFERENTIAL GAIN TERM INTO
012B 0034               MOVWF   mulplr          ; MULPLR TO MULTIPLY THE DE/DT
012C 0901               CALL    mpy_S           ; DO THE MULTIPLICATION
012D 091D               CALL    DIV_LMT         ; SCALE AND LIMIT TO 100

                RE_SGN
012E 0759               BTFSS   FLAGS,DE_SGN    ; IF THE DE SIGN IS NEGATIVE THEN
012F 0B32               GOTO    SAVE_DIFF       ; PUT THE SIGN INTO THE LOW BYTE
0130 0276               COMF    L_byte,1
0131 02B6               INCF    L_byte,1
                SAVE_DIFF
0132 0216               MOVF    L_byte,W
0133 0643               BTFSC   SWR,Z
0134 0B45               GOTO    ROLL_ER
0135 002F               MOVWF   ERR_O

                ; ADD THE DIFF TERM INTO THE SUMM ***************
                ADDDIF
0136 0C00               MOVLW   00
0137 0659               BTFSC   FLAGS,DE_SGN    ; PUT THE KD*(DE/DT) TERM INTO THE
0138 0CFF               MOVLW   0FF             ; REGISTERS TO ADD. AND
0139 0036               MOVWF   ACCbHI          ; SIGN EXTEND THE UPPER BYTE
013A 020F               MOVF    ERR_O,W
013B 0034               MOVWF   ACCbLO
013C 020D               MOVF    SUMLO,W         ; LOAD THE CURRENT SUM INTO THE
013D 0033               MOVWF   ACCaLO          ; REGISTERS TO ADD
013E 0218               MOVF    SUMHI,W
013F 0035               MOVWF   ACCaHI
0140 0910               CALL    D_add           ; ADD IN THE DIFFERENTIAL TERM
0141 0214               MOVF    ACCbLO,W        ; SAVE THE RESULTS BACK
0142 002D               MOVWF   SUMLO           ; INTO SUMLO AND HI
0143 0216               MOVF    ACCbHI,W
0144 0038               MOVWF   SUMHI

                ROLL_ER
0145 020C               MOVF    ERROR,W         ; TAKE THE CURRENT ERROR
0146 002F               MOVWF   ERR_O           ; AND PUT IT IN THE ERROR HISTORY
0147 0499               BCF     FLAGS,OER_SGN   ; SAVE THE CURRENT ERROR SIGN
0148 0619               BTFSC   FLAGS,ER_SGN    ; IN THE OLD ERROR SIGN FOR
0149 0599               BSF     FLAGS,OER_SGN   ; NEXT TIME THROUGH


                ;****SET UP THE DIRECTION FOR THE BRIDGE*********************
                ;
                ; After the sum of all the components has been made, the sign of the
                ; sum will determine which way the bridge should be powered.
                ; If the sum is negative the bridge needs to be set to drive ccw; if the
                ; sum is posative then the bridge needs to be set to drive cw.  This
                ; is purely a convention and depends upon the polarity the motor and feedback
                ; element are hooked up in.
                SET_DIR
014A 0479               BCF     FLAGS,DIR       ; SET FOR DEFAULT CLOCKWISE
014B 06F8               BTFSC   SUMHI,7         ; LOOK AT THE SIGN BIT, IF IT IS SET
014C 0579               BSF     FLAGS,DIR       ; THEN SET FOR CCW BRIDGE DRIVE

                ;**** SCALE THE NUMBER TO BETWEEN 0 AND 100% **********************
                ; After the direction is set the request for duty cycle is limited to between
                ; 0 and 100 percent inclusive.  This value is passed to the dutycycle setting
                ; routine by loading it in the variable "PCNT".
```


© 1994 Microchip Technology Incorporated

```
                        L_SUMM
014D 07F8                       BTFSS   SUMHI,7         ; CHECK TO SEE IF IT IS NEGATIVE
014E 0B52                       GOTO    POS_LM
014F 0278                       COMF    SUMHI,1
0150 026D                       COMF    SUMLO,1
0151 02AD                       INCF    SUMLO,1

                        POS_LM
0152 0C01                       MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0153 0098                       SUBWF   SUMHI,0         ; IF THERE IS ANYTHING THERE, IF NOT,
0154 0703                       BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0155 0B59                       GOTO    LB_L            ; OTHERWISE GIVE THE LOW BYTE A FULL
0156 0C64                       MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
0157 002D                       MOVWF   SUMLO           ; TO 100
0158 0B5F                       GOTO    LP_EXIT         ; GOTO LIMIT PERCENT EXIT
                        LB_L
0159 0C64                       MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
015A 008D                       SUBWF   SUMLO,0         ; 100 DECIMAL
015B 0703                       BTFSS   SWR,CARRY
015C 0B5F                       GOTO    LP_EXIT
015D 0C64                       MOVLW   64H
015E 002D                       MOVWF   SUMLO

                        LP_EXIT
015F 020D                       MOVF    SUMLO,W         ; STORE THE LIMITED VALUE IN
0160 0029                       MOVWF   PCNT            ; THE PERCENT DUTYCYCLE REQUEST


                        ;*******************************************************
                        ; PWM GENERATING ROUTINE
                        ;
                        ; The important thing here is not to have to do too many decisions or
                        ; calculations while you are generating the 100 or so pulses.  These will
                        ; take time and limit the minimum or maximum duty cycle.

                        WHICH_DIR
0161 0679                       BTFSC   FLAGS,DIR       ; CHECK THE DIRECTION FLAG
0162 0B76                       GOTO    GOCCW           ; DO CCW PULSES FOR 1
0163 0B64                       GOTO    GOCW            ; DO CW PULSES FOR 0


                        GOCW
0164 0426                       BCF     PORTB,PWMCCW    ; SET THE BRIDGE FOR CW MOVE
0165 0C64                       MOVLW   64H             ;
0166 0032                       MOVWF   CYCLES          ; SET UP CYCLES COUNTER FOR 100 PULSES
0167 0943                       CALL    CALCTIMES       ; CALCULATE THE HI AND LO TIMES

                        RLDCW
0168 0207                       MOVF    HI,0            ; RE LOAD THE HI TIMER
0169 002A                       MOVWF   HI_T            ; WITH THE CALCULATED TIME
016A 0208                       MOVF    LO,0            ; RE LOAD THE LO TIMER
016B 002B                       MOVWF   LO_T            ; WITH THE CALCULATED TIME
016C 0004                       CLRWDT                  ; TAG THE WATCHDOG TIMER

                        CWHI
016D 0506                       BSF     PORTB,PWMCW     ; SET THE CLOCKWISE PWMBIT HIGH
016E 02EA                       DECFSZ  HI_T,1          ; DECREMENT THE HI USEC. COUNTER
016F 0B6D                       GOTO    CWHI            ; DO ANOTHER LOOP
                        CWLO
0170 0406                       BCF     PORTB,PWMCW     ; SET THE CLOCKWISE PWM BIT LOW
0171 02EB                       DECFSZ  LO_T,1          ; DECREMENT THE LO USEC. COUNTER
0172 0B70                       GOTO    CWLO            ; DO ANOTHER LOOP
0173 02F2                       DECFSZ  CYCLES,1        ; DECREMENT THE NUMBER OF CYCLES LEFT
0174 0B68                       GOTO    RLDCW           ; DO ANOTHER PULSE
0175 0A50                       GOTO    BEGIN           ; DO ANOTHER MAIN SYSTEM CYCLE
```


© 1994 Microchip Technology Incorporated

```
                    GOCCW
0176 0406                   BCF     PORTB,PWMCW     ; SET THE BRIDGE FOR CCW MOVE
0177 0C64                   MOVLW   64H             ;
0178 0032                   MOVWF   CYCLES          ; SET UP CYCLE COUNTER FOR 100 PULSES
0179 0943                   CALL    CALCTIMES       ; CALCULATE THE HI AND LO TIMES
                    RLDCCW
017A 0207                   MOVF    HI,0            ; RE LOAD THE HI TIMER
017B 002A                   MOVWF   HI_T            ; WITH THE CALCULATED TIME
017C 0208                   MOVF    LO,0            ; RE LOAD THE LO TIMER
017D 002B                   MOVWF   LO_T            ; WITH THE CALCULATED TIME
017E 0004                   CLRWDT                  ; TAG THE WATCHDOG
                    CCWHI
017F 0526                   BSF     PORTB,PWMCCW    ; SET THE COUNTERCLOCKWISE PWM BIT HIGH
0180 02EA                   DECFSZ  HI_T,1          ; DECREMENT THE HI USEC. COUNTER
0181 0B7F                   GOTO    CCWHI           ; DO ANOTHER LOOP
                    CCWLO
0182 0426                   BCF     PORTB,PWMCCW    ; SET THE COUNTERCLOCKWISE PWM BIT LOW
0183 02EB                   DECFSZ  LO_T,1          ; DECREMENT THE LO USEC. COUNTER
0184 0B82                   GOTO    CCWLO           ; DO ANOTHER LOOP
0185 02F2                   DECFSZ  CYCLES,1        ; DECREMENT THE NUMBER OF CYCLES LEFT
0186 0B7A                   GOTO    RLDCCW          ; DO ANOTHER PULSE
0187 0A50                   GOTO    BEGIN           ; DO ANOTHER MAIN SYSTEM CYCLE



                ;************* START VECTOR ********************

                    CLRREG                      ;INITIALIZE REGISTERS
0188 0C0B                   MOVLW   0BH             ; SET PORT A FOR 3 INPUTS AND
0189 0005                   TRIS    PORTA           ; AN OUTPUT
018A 0C1C                   MOVLW   1CH             ; SET PORT B FOR INPUTS AND OUTPUTS
018B 0006                   TRIS    PORTB           ; THIS SETTING FOR SENDING TO A/D
018C 0040                   CLRW                    ; CLEAR THE W REGISTER
018D 0002                   OPTION                  ; STORE THE W REG IN THE OPTION REG
018E 0C08                   MOVLW   08H             ; STARTING REGISTER TO ZERO
018F 0024                   MOVWF   FSR             ;
                    GCLR
0190 0060                   CLRF    00              ;
0191 03E4                   INCFSZ  FSR             ; SKIP AFTER ALL REGISTERS
0192 0B90                   GOTO    GCLR            ; HAVE BEEN INITIALIZED
0193 0A50                   GOTO    BEGIN           ; START AT THE BEGINING OF THE PROGRAM

                            ORG     01FF            ;
01FF 0B88                   GOTO    CLRREG          ; START VECTOR

                          END


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Incorporated

A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs

INTRODUCTION

The purpose of this application note is to design a clock while multiplexing the features as much as possible, allowing the circuit to use the 18-pin PIC16C54. Other devices in the Microchip Technology Inc. line expand on this part, making it a good starting point for learning the basics. This design is useful because it utilizes every pin for output and switches some of them to inputs briefly to read the keys. For a more extensive clock design, consult application note AN529.

THE DESIGN

This design is a simple time of day clock incorporating four seven-segment LED displays and three input switches. There is also an additional reset switch that would not normally be incorporated into the final design. The schematic is illustrated in Figure 1.

CONNECTIONS

The individual segments of each display are connected together, A-A-A-A, B-B-B-B, etc. The displays are numbered from the right, or least significant digit. The second display from the right is flipped upside down to align its decimal with the third display, creating the center clock colon. Therefore the segments are not tied together evenly straight across on the board, but must compensate for the change in one display's orientation. The common cathode for each display is turned on with transistors connected to the four I/O lines of Port A. The connections are RA0-CC4/Digit4, RA1-CC3/Digit3, RA2-CC2/Digit2, RA3-CC1/Digit1. A low output turns on the PNP transistor for the selected display. The Port B pins activate the LED segments. For this design only the center colon decimal points were connected. The connections are RB0-dp, RB1-A, RB2-B, RB3-C...RB7-G.

The switches are also connected to Port B I/O pins. Port B pins RB1, RB2, and RB3 are pulled low with 10K ohm resistors. This value is high enough to not draw current away from the LEDs when they are being driven on. Inputs are detected by pulling the pins high with a switch to V_{DD} through 820 ohm resistors. This value is low enough to pull the pin high quickly when the outputs have been turned off, and to create a 90% of V_{DD} high input.

OPERATION

Switches

When no buttons are pressed, the circuit will display the current time, starting at 12:00 on reset. Pressing SW1 will cause seconds to be displayed. The time is set by pressing SW2 to advance minutes, and SW3 to advance hours. Since each of the segments are tied together across all displays, only one display should be turned on at a time, or all displays turned on would display duplicate data. The displays are turned on right to left, with each display's value being output its turn. This is done fast enough so that there is no perceived flicker. The switches are read between display cycles.

Timing

The PIC16CXX prescaler is assigned to the RTCC as a 1:16 divide. The RTCC pin is tied low since it is not used. The OPTION Register is loaded with 03h to initialize this prescaler set up. The software is written with timing based on a 4.000mhz crystal. The instruction clock is 1.000 MHz after the internal divide by four. The 8-bit RTCC register rolls over every 256 cycles, for a final frequency of 244.1406 Hz. (exactly a 4.096 ms period) A variable named sec_nth is used to count 244 roll-overs of the RTCC for one second. The benefit of keeping time with a nth variable is that it can be written to as needed to adjust time in "nths" of a second, allowing almost any odd crystal frequency to be used. Simply determine the best prescale and "nth" divider, and compute the "nth" adjustment needed for each minute, hour, twelve hour roll-over. Time can be kept accurately to two "nths" a day (an "nth" is 1/244 of a second in this case). In this circuit, 9 "nths" are subtracted each minute, 34 "nths" are added each hour, and 6 "nths" subtracted every twelve hour roll-over. This leaves a computed error of 1.5 seconds/year except for crystal frequency drift. Another possible solution is to initialize the RTCC to some value that causes a roll-over at a predetermined time interval. Writing to the RTCC causes two clock cycles to be missed while clock edges realign, which would have to be accounted for. This is described in the *Microchip Data Book*.

© 1994 Microchip Technology Inc.

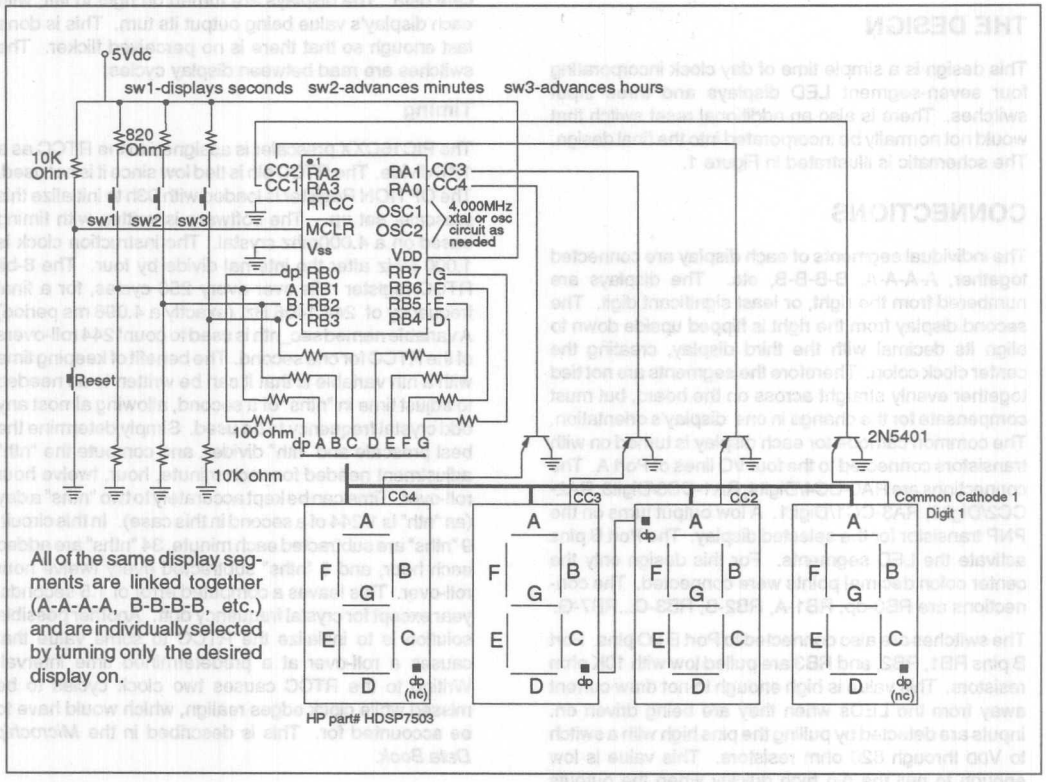
Displays

The program contains portions of code that act as a display driver. A variable exists for each of the four displays. A hex value from 0 to 9 can be written to these variables and they will be converted to display code and output to the displays. Only one display is actually on at a time, and its code is output into it in its turn. Another section of code takes the seconds, minutes, or hours value and separates it into the two digits needed for each display. In other words, 48 seconds would be separated into a "4" and an "8" and written to the appropriate display variable. The displays used were common cathode and turned on with transistors to avoid trying to sink too much current into the PIC16CXX. A display is enabled with a zero at the appropriate pin. 100 ohm resistors were used in series with the segments to obtain the desired brightness. Different values may be required if different displays are used. Since the displays are each on less than one fourth of the time, the resistor value must be low enough to compensate for the needed forward current.

CONCLUSION

The instruction execution speed of the PIC16C54 (and the rest of the PIC16/17 series) allows many functions to be implemented on a few pins by multiplexing them in software. User inputs, Real Time Clock Counter, and multiple LED displays are all accommodated with little or no sacrifice in functionality.

FIGURE 1: TIME OF DAY CLOCK USING PIC16C54



Author: Dan Matthews
Corporate Applications Manager

© 1994 Microchip Technology Inc.

```
-LOC   LINE SOURCE TEXT

       0001  list P = 16C54
       0002 ;
       0003 ;*****************************************************************
       0004 ;                          Clock
       0005 ;*****************************************************************
       0006 ;
       0007 ;                   PROGRAM DESCRIPTION
       0008 ;
       0009 ; This program runs on a PIC16C54.
       0010 ;
       0011 ;                          Hardware Description
       0012 ;
       0013 ;  DISPLAYS
       0014 ; Four 7 segment displays are multiplexed.  The segments are tied together,
       0015 ; with the common cathode pins broken out separately.  The display appears
       0016 ; as a clock with a center semicolon ( 88:88 ).  The segments are assigned
       0017 ; to Port B, with the semicolon being RB0, and segments A through F
       0018 ; assigned as RB1 to RB7 respectively.
       0019 ;  The four common cathodes are activated by the four Port A pins through
       0020 ; transistors. RA0 for Digit4, RA1/Digit3, RA2/Digit2... through Digit4,
       0021 ; with Digit1 being in the rightmost position. The center semicolon is
       0022 ; made from the decimals of LED 2 and 3.
       0023 ;  Digit2 is turned upside down to put its decimal into position,
       0024 ; but it is wired with a corrected A-F assignment to compensate.  Both
       0025 ; decimals are tied together at RB0, but the display cathodes are still
       0026 ; separate.  Activating the decimal for digit2 AND 3 will turn on the
       0027 ; center colon.
       0028 ;
       0029 ;  SWITCHES
       0030 ; Because all twelve I/O pins are already used for the muxed displays,
       0031 ; eight for segments and four for digit selection, the three switches must
       0032 ; be read alternatingly through software.  The switches lie
       0033 ; across Port B pins, which are changed to inputs momentarily during read
       0034 ; and changed back to outputs during display.  Enough series resistance
       0035 ; must be used to prevent turning on or shorting segments during display
       0036 ; cycles if a switch is pressed.
       0037 ;
       0038 ; SW1-displays seconds, SW2-advances minutes,
       0039 ; SW3-advances hours, (none)-displays time
       0040 ;
       0041 ;*************************  Header ************************
       0042 ;
       0043 ;
01FF   0044 PIC54       equ     H'01FF' ; start address if used in a PIC16C54
03FF   0045 PIC56       equ     H'03FF' ;   "     "     "    "  " PIC16C56
       0046                            ;
0000   0047 POINTER     equ     H'00'   ; address location f0 is an indirect address pointer
0001   0048 RTCC        equ     H'01'   ; address of RTCC clock value
0002   0049 PC          equ     H'02'   ; program counter
0003   0050 STATUS      equ     H'03'   ; F3 Reg is STATUS Reg.
0004   0051 FSR         equ     H'04'   ; F4 is File Select Register, address POINTER will direct to.
       0052                            ;
0005   0053 PORT_A      equ     H'05'   ; 7 segment Display Common Cathodes
0006   0054 PORT_B      equ     H'06'   ; Muxed Display Segments (Switches when inputs)
       0055                            ;
       0056                            ; STATUS REG. Bits
0000   0057 CARRY       equ     0       ; Carry Bit is Bit.0 of F3
0000   0058 C           equ     0
0001   0059 DCARRY      equ     1
0001   0060 DC          equ     1
0002   0061 Z_bit       equ     2       ; Bit 2 of F3 is Zero Bit
0002   0062 Z           equ     2
0003   0063 P_DOWN      equ     3
0003   0064 PD          equ     3
0004   0065 T_OUT       equ     4
0004   0066 TO          equ     4
0005   0067 PA0         equ     5       ;16C5X Status bits
```

© 1994 Microchip Technology Inc.

```
0006    0068 PA1        equ     6           ;16C5X Status bits
0007    0069 PA2        equ     7           ;16C5X Status bits
        0070                                ;
007E    0071 ZERO       equ     H'7E'
000C    0072 ONE        equ     H'0C'
00B6    0073 TWO        equ     H'B6'
009E    0074 THREE      equ     H'9E'
00CC    0075 FOUR       equ     H'CC'
00DA    0076 FIVE       equ     H'DA'
00FA    0077 SIX        equ     H'FA'       ; coding of segments for display (PORT_B)
000E    0078 SEVEN      equ     H'0E'
00FE    0079 EIGHT      equ     H'FE'
00CE    0080 NINE       equ     H'CE'
0000    0081 BLANK      equ     H'00'
        0082                                ;
        0083                                ; timer variables start at a number that allows
        0084                                ; rollover in sync with time rollover, i.e. seconds
        0085                                ; starts at decimal 196 so that sixty 1-second
                                            ;  increments causes 0.
000C    0086 MAXNTHS    equ     D'12'       ;  initialization constants for timer count up
00C4    0087 MAXSECS    equ     D'196'      ;     see variable explanations for more info
00C4    0088 MAXMINS    equ     D'196'      ;
00F4    0089 MAXHRS     equ     D'244'
00F3    0090 MINHRS     equ     D'243'
0009    0091 ADJMIN     equ     D'9'        ; number of nths to be subtracted each minute for
                                            ;  accuracy
0022    0092 ADJHR      equ     D'34'       ; nths added each hour for accurate time
0006    0093 ADJDAY     equ     D'6'        ; nths subtracted each 1/2 day rollover
        0094 ;
00FE    0095 DISP4      equ     B'11111110'
00FD    0096 DISP3      equ     B'11111101' ; Mapping of Active Display Selection (PORT_A)
00FB    0097 DISP2      equ     B'11111011' ;  displays are active low
00F7    0098 DISP1      equ     B'11110111'
00FF    0099 DISPOFF    equ     H'FF'       ; turns all displays off when written to PORT_A
000E    0100 SWITCH     equ     B'00001110' ; Used in tris B to set RB1-3 for switch inputs
        0101                                ;
        0102                                ;  Flag bit assignments
0000    0103 SEC        equ     H'0'        ; update time display values for sec, min, or hours
0001    0104 MIN        equ     H'1'
0002    0105 HRS        equ     H'2'
0003    0106 CHG        equ     H'3'        ; a change has occurred on a switch or a display
                                            ;   value
0004    0107 SW1        equ     H'4'        ; Flag bit assignments - switches that are on = 1
0005    0108 SW2        equ     H'5'        ;  SW1 is Seconds-minutes, SW2-hours, SW3-mode
0006    0109 SW3        equ     H'6'
0007    0110 SW_ON      equ     H'7'        ; indicates a switch has been pressed
        0111                                ;
        0112                                ; RAM VARIABLES
        0113 ;          equ     H'08'       ; not used
0009    0114 flags      equ     H'09'       ; bits:0-SEC,1-MIN,2-HRS,3-CHG,4-SW1,5-SW2,6-SW3,7-SW_ON
000B    0115 display    equ     H'0B'       ; SW_ON variable location - which display to update
000C    0116 digit1     equ     H'0C'       ; Rightmost display value
000D    0117 digit2     equ     H'0D'       ; Second display from right
000E    0118 digit3     equ     H'0E'       ; Third    "      "     "
000F    0119 digit4     equ     H'0F'       ; Fourth (and Leftmost)
0010    0120 sec_nth    equ     H'10'       ; seconds, fractional place
0011    0121 seconds    equ     H'11'       ; seconds
0012    0122 minutes    equ     H'12'       ; minutes
0013    0123 hours      equ     H'13'       ; hours
0014    0124 var        equ     H'14'       ; variable for misc math computations
0015    025  count      equ     H'15'       ; loop counter variable
        0126                                ;
        0127                                ;*****************************************
        0128                                ;
        0129                                ;  Initialize Ports all outputs, blank display
        030                                 ;
```


© 1994 Microchip Technology Inc.

```
      OBJECT
-LOC  CODE  LINE SOURCE TEXT
0000  0C03  0131 START   movlw   H'03'      ; set option register, transition on clock,
0001  0002  0132          option              ; Prescale RTCC, 1:16
                0133                              ;
0002  0C00  0134          movlw   0
0003  0005  0135          tris    PORT_A      ; Set all port pins as outputs
0004  0006  0136          tris    PORT_B
0005  0C00  0137          movlw   BLANK
0006  0026  0138          movwf   PORT_B      ; Blank the display
0007  04C3  0139          bcf     STATUS,PA1  ; page zero in case this is a higher PIC version
0008  04A3  0140          bcf     STATUS,PA0
              0141                              ;
              0142                              ; initialize variables
0009  0C01  0143          movlw   H'01'
000A  0021  0144          movwf   RTCC        ; set RTCC above zero so initial wait period occurs
000B  0CF7  0145          movlw   DISP1
000C  002B  0146          movwf   display     ; initializes display selected to first display.
000D  0C00  0147          movlw   BLANK       ; put all displays to blank, no visible segments
000E  002C  0148          movwf   digit1
000F  002D  0149          movwf   digit2
0010  002E  0150          movwf   digit3
0011  002F  0151          movwf   digit4
0012  0C0C  0152          movlw   MAXNTHS     ; set timer variables to initial values
0013  0030  0153          movwf   sec_nth
0014  0CC4  0154          movlw   MAXSECS
0015  0031  0155          movwf   seconds
0016  0CC4  0156          movlw   MAXMINS
0017  0032  0157          movwf   minutes
0018  0CFF  0158          movlw   H'FF'       ; hours start at 12 which is max at FF
0019  0033  0159          movwf   hours
001A  0C00  0160          movlw   H'00'
001B  0029  0161          movwf   flags       ; clear the flags variable
001C  0004  0162          clrwdt              ; clear WatchDog Timer, must be within every 18ms
              0163                              ;
              0164                              ;
              0165 MAIN
              0166                              ;
              0167                              ; wait for RTCC to roll-over
              0168 RTCC_FILL
001D  0201  0169          movf    RTCC,0
001E  0743  0170          btfss   STATUS,Z    ; note, RTCC is left free running
001F  0A1D  0171          goto    RTCC_FILL
              0172                              ;
0020  03F0  0173          incfsz  sec_nth,1   ;add 1 to nths, n X nths = 1 sec, n is based on pres
0021  0A54  0174          goto    TIME_DONE
0022  0004  0175          clrwdt
0023  0C0C  0176          movlw   MAXNTHS
0024  0030  0177          movwf   sec_nth     ; restore sec_nths variable for next round
              0178                              ;
              0179 CHECK_SW
0025  07E9  0180          btfss   flags,SW_ON ; if no switches pressed, bypass this
0026  0A3C  0181          goto    SET_TIME
0027  0689  0182          btfsc   flags,SW1
0028  0A3C  0183          goto    SET_TIME    ; if seconds display is pressed, do not change time
0029  0CC4  0184          movlw   MAXSECS
002A  0031  0185          movwf   seconds     ; reset seconds to zero when setting clock
002B  0C7F  0186          movlw   H'7F'
002C  0030  0187          movwf   sec_nth     ; advance second timer 1/2 second to speed setting
002D  07A9  0188          btfss   flags,SW2
002E  0A35  0189          goto    HOURSET     ; if minutes do not need changing, check hours
002F  0CAF  0190          movlw   H'AF'
0030  0030  0191          movwf   sec_nth     ; advances timer faster when setting minutes
0031  03F2  0192          incfsz  minutes,1   ; advances minutes 1
0032  0A35  0193          goto    HOURSET
0033  0CC4  0194          movlw   MAXMINS
0034   0032 0195          movwf   minutes     ; if minutes roll over to zero, reinitialize
                                              ;    minutes
              0196                              ;
```

© 1994 Microchip Technology Inc.

```
                        incfsz  hours,1          ; advance hours 1
0038  0A60  0200        goto    CHECK_TIME
0039  0CF4  0201        movlw   MAXHRS          ; if hours rolls over to zero, reinitialize
003A  0033  0202        movwf   hours
003B  0A60  0203        goto    CHECK_TIME      ;  skip time keeping, go to display changes
            0204                                ;
            0205 SET_TIME
003C  0509  0206        bsf     flags,SEC       ; indicates seconds, if displayed, should be
                                                ;   updated
003D  0569  0207        bsf     flags,CHG       ; inicates a flag change was made.
003E  03F1  0208        incfsz  seconds,1       ;  add 1 to seconds
003F  0A54  0209        goto    TIME_DONE
0040  0CC4  0210        movlw   MAXSECS
0041  0031  0211        movwf   seconds         ; restore seconds variable for next round
            0212                                ;
0042  0529  0213        bsf     flags,MIN       ; minutes, if displayed, should be updated
0043  0569  0214        bsf     flags,CHG       ; indicates a flag change was made
0044  0C09  0215        movlw   ADJMIN
0045  00B0  0216        subwf   sec_nth,1       ; accuracy adjustment, do not go below 0
0046  03F2  0217        incfsz  minutes,1       ; add 1 to minutes
0047  0A54  0218        goto    TIME_DONE
0048  0CC4  0219        movlw   MAXMINS
0049  0032  0220        movwf   minutes         ; restore minutes variable for next hour
                                                ;   countdown
            0221                                ;
004A  0549  0222        bsf     flags,HRS       ; hours, if displayed, should be updated
004B  0569  0223        bsf     flags,CHG       ; indicates a flag change was made
004C  0C22  0224        movlw   ADJHR
004D  01F0  0225        addwf   sec_nth,1       ; add needed adjustment to nths for each hour
004E  03F3  0226        incfsz  hours,1         ; add 1 to hours
004F  0A54  0227        goto    TIME_DONE
0050  0CF4  0228        movlw   MAXHRS
0051  0033  0229        movwf   hours           ; restore hours variable for next round
0052  0C06  0230        movlw   ADJDAY
0053  00B0  0231        subwf   sec_nth,1       ; subtraction adjustment for each 1/2 day rollover
            0232                                ;
            0233 TIME_DONE
0054  0769  0234        btfss   flags,CHG       ; if no switches or potentially dislayed
                                                ;   numbers
0055  0A91  0235        goto    CYCLE           ;  were changed, then leave the display same
            0236                                ;
            0237                                ;
            0238 CHECK_SECONDS
            0239                                ;  if seconds button was pushed display
                                                ;   seconds
0056  0789  0240        btfss   flags,SW1
0057  0A60  0241        goto    CHECK_TIME      ; if seconds button not pressed, skip this
0058  0C00  0242        movlw   H'00'           ; zero time display variables except seconds
                                                ;   (digit1)
0059  002D  0243        movwf   digit2          ; digit1 used to temporarily hold hex seconds
                                                ;   or minutes
005A  002E  0244        movwf   digit3
005B  002F  0245        movwf   digit4
005C  0CC4  0246        movlw   MAXSECS
005D  0091  0247        subwf   seconds,0       ; subtract initialized preset to get actual
                                                ;   seconds
005E  002C  0248        movwf   digit1          ; 1st digit variable temporarily holds hex
                                                ;   value seconds
005F  0A69  0249        goto    SPLIT_HEX       ; done updating display variables in hex
            0250                                ;
            0251 CHECK_TIME
0060  0C00  0252        movlw   H'00'
0061  002F  0253        movwf   digit4          ; zero out tens places in case there is no tens
                                                ;   increment
0062  002D  0254        movwf   digit2
```


© 1994 Microchip Technology Inc.

```
0063  0CF3   0255          movlw   MINHRS
0064  0093   0256          subwf   hours,0          ; subtract initialized preset to get actual
                                                    ;   hours
0065  002E   0257          movwf   digit3           ; 3rd digit variable temporarily holds hex
                                                    ;   value for hours
0066  0CC4   0258          movlw   MAXMINS
0067  0092   0259          subwf   minutes,0        ; subtract initialized preset to get actual
                                                    ;   minutes
0068  002C   0260          movwf   digit1           ; 1st digit temporarily holds hex value for
                                                    ;   minutes
              0261                                  ;
              0262                                  ; note digit variables are used for temp
                                                    ;   variables and final display variables
              0263                                  ;
              0264 SPLIT_HEX                        ;  split into two hex display variables and
                                                    ;  write
              0265                                  ;
0069  0C02   0266          movlw   H'02'
006A  0035   0267          movwf   count            ; convert each number - seconds - or minutes
                                                    ;   and hours
              0268
              0269                                  ;1st time through, FSR = digit1, 2nd time FSR =
                                                    ;   digit3
006B  0C0C   0270          movlw   digit1
006C  0024   0271          movwf   FSR              ; address of digit1 into File Select Register
                                                    ;   prepares POINTER
006D  0A70   0272          goto    LOOP             ; this loop is used to modify the minutes/
                                                    ;   seconds place
              0273                                  ;
006E  0C0E   0274 LOOP2    movlw   digit3
006F  0024   0275          movwf   FSR              ; this loop is used to modify the hours place
              0276                                  ;
              0277 LOOP
0070  0C0A   0278          movlw   D'10'
0071  00A0   0279          subwf   POINTER,1        ; find out how many tens in number,
0072  0603   0280          btfsc   STATUS,C         ; was a borrow needed?
0073  0A76   0281          goto    INCREMENT_10S    ;  if not, add 1 to tens position
0074  01E0   0282          addwf   POINTER,1        ;  if so, do not increment tens place, add ten
                                                    ;   back on
0075  0A7A   0283          goto    NEXT_DIGIT
              0284                                  ;
              0285 INCREMENT_10S
0076  02A4   0286          incf    FSR,1            ; bump address pointed to from 1s positoion to
                                                    ;   10s
0077  02A0   0287          incf    POINTER,1        ; add 1 to 10s position as determined by
                                                    ;   previous subtract
0078  00E4   0288          decf    FSR,1            ; put POINTER value back to 1s place for next
                                                    ;   subtraction
0079  0A70   0289          goto    LOOP             ; go back and keep subtracting until finished
              0290                                  ;
              0291 NEXT_DIGIT
007A  02F5   0292          decfsz  count,1
007B  0A6E   0293          goto    LOOP2            ; after splitting minutes into two places, go
                                                    ;   split hours
              0294                                  ;
              0295 CONVERT_HEX_TO_DISPLAY           ; converts digit variables to decimal display
                                                    ;   code
007C  0C0C   0296          movlw   digit1
007D  0024   0297          movwf   FSR              ; put the address of the digit1 into the FSR to
                                                    ;   enable POINTER
007E  0C04   0298          movlw   H'04'
007F  0035   0299          movwf   count            ; prepare count variable to loop for all four
                                                    ;   displays
              0300 NEXT_HEX
0080  0200   0301          movf    POINTER,0        ; get the hex value of the current digit vari
                                                    ;   able
0081  09C3   0302          call    RETURN_CODE      ; call for the hex to segment display code
                                                    ;   conversion
0082  0020   0303          movwf   POINTER          ; put the returned display code into the digit
```

© 1994 Microchip Technology Inc.

```
                                                    ;   variable
0083  02A4   0304          incf    FSR,1            ; increment the pointer to the next digit
                                                    ;   address
0084  02F5   0305          decfsz  count,1          ; allow only count(4) times through loop
0085  0A80   0306          goto    NEXT_HEX
             0307                                   ;
             0308 FIX_DISPLAY
0086  0C7E   0309          movlw   ZERO
0087  008F   0310          subwf   digit4,0         ; check to see if left digit is a zero, if so
                                                    ;   blank it out
0088  0743   0311          btfss   STATUS,Z
0089  0A8C   0312          goto    FIX_SEC
008A  0C00   0313          movlw   BLANK
008B  002F   0314          movwf   digit4
             0315
008C  0789   0316 FIX_SEC  btfss   flags,SW1        ; if seconds are displayed, blank the third
                                                    ;   display too
008D  0A8F   0317          goto    CLEAR_FLAGS
008E  002E   0318          movwf   digit3
             0319                                   ;
             0320 CLEAR_FLAGS
008F  0CF0   0321          movlw   H'F0'
0090  0169   0322          andwf   flags,1          ; clear the lower 4 flag bits to show updated
                                                    ;   time status
             0323                                   ;
             0324 CYCLE
0091  0CFF   0325          movlw   DISPOFF
0092  0025   0326          movwf   PORT_A           ; Turn off LED Displays
0093  0C0E   0327          movlw   SWITCH
0094  0006   0328          tris    PORT_B           ; Set some port B pins as switch inputs
0095  0C0F   0329          movlw   H'0F'
0096  0169   0330          andwf   flags,1          ; reset switch flags to zero
0097  0000   0331          nop                      ; nop may not be needed, allows old outputs to
0098  0000   0332          nop                      ;   bleedoff through 10k R before reading port
                                                    ;   pins
0099  0000   0333          nop
009A  0206   0334          movf    PORT_B,0         ; read PORT_B for switch status
009B  0034   0335          movwf   var              ;  write switch status to temporary variable
                                                    ;   "var"
009C  0734   0336          btfss   var,1
009D  0AA1   0337          goto    SWITCH2          ; indicate which switches are pressed in the
                                                    ;   flags variable
009E  0569   0338          bsf     flags,CHG
009F  0589   0339          bsf     flags,SW1
00A0  05E9   0340          bsf     flags,SW_ON
00A1  0754   0341 SWITCH2  btfss   var,2
00A2  0AA6   0342          goto    SWITCH3
00A3  0569   0343          bsf     flags,CHG
00A4  05A9   0344          bsf     flags,SW2
00A5  05E9   0345          bsf     flags,SW_ON
00A6  0774   0346 SWITCH3  btfss   var,3
00A7  0AAB   0347          goto    SETPORT
00A8  0569   0348          bsf     flags,CHG
00A9  05C9   0349          bsf     flags,SW3
00AA  05E9   0350          bsf     flags,SW_ON
             0351                                   ;
00AB  0C00   0352 SETPORT  movlw   H'00'            ; restore PORT_B as all outputs to displays
00AC  0006   0353          tris    PORT_B
00AD  0C00   0354          movlw   BLANK            ; blank display in preparation for next digit
                                                    ;   cycle
00AE  0026   0355          movwf   PORT_B
             0356                                   ;
             0357                                   ; determine which display needs updating and
                                                    ;   cycle it on
00AF  070B   0358          btfss   display,0        ; if 1st display, get 1st digit value into w
00B0  020F   0359          movf    digit4,0
00B1  072B   0360          btfss   display,1        ; if 2nd display, get 2nd digit
00B2  020E   0361          movf    digit3,0
00B3  074B   0362          btfss   display,2        ; if 3rd display, get 3rd digit
```


© 1994 Microchip Technology Inc.

```
00B4  020D   0363            movf    digit2,0
00B5  076B   0364            btfss   display,3       ; if 4th display, get 4th digit
00B6  020C   0365            movf    digit1,0
00B7  0026   0366            movwf   PORT_B          ; put the number in w out to display
00B8  06F0   0367            btfsc   sec_nth,7
00B9  0506   0368            bsf     PORT_B,0        ; sets colon decimal on %50 duty using highest
                                                     ;   bit
00BA  020B   0369            movf    display,0       ; get display needing cycle on
00BB  0025   0370            movwf   PORT_A          ; enables proper display
00BC  002B   0371            movwf   display         ; enables display selected in last pass of
                                                     ;   CYCLE
00BD  036B   0372            rlf     display,1       ; rotate display "on" bit to next position
00BE  050B   0373            bsf     display,0       ; assures a 1 on lowest position since rotated
                                                     ;   in carry is zero
00BF  078B   0374            btfss   display,4       ; check if last display was already updated
00C0  040B   0375            bcf     display,0       ; if it was, set display back to 1st (bit 0
                                                     ;   cleared)
00C1  0004   0376            clrwdt                  ; this program pass completed normally, reset
                                                     ;   watch dog
             0377                                    ;
             0378                                    ;
             0379                                    ;
00C2  0A1D   0380            goto    MAIN
             0381                                    ;
             0382 RETURN_CODE
             0383                                    ;
00C3  01E2   0384            addwf   PC,1            ; the hex value in the display variable is
00C4  087E   0385            retlw   ZERO            ;   added to PC which causes a jump to return
                                                     ;   its display code
00C5  080C   0386            retlw   ONE
00C6  08B6   0387            retlw   TWO
00C7  089E   0388            retlw   THREE
00C8  08CC   0389            retlw   FOUR
00C9  08DA   0390            retlw   FIVE
00CA  08FA   0391            retlw   SIX
00CB  080E   0392            retlw   SEVEN
00CC  08FE   0393            retlw   EIGHT
00CD  08CE   0394            retlw   NINE
             0395                                    ;
             0396                                    ;
             0397            org     PIC54           ; reset location for this processor
Warning: Crossing page boundary — ensure page bits are set
01FF  0A00   0398            goto    START           ; begin program execution at START label
             0399                                    ;
             0400     END
             0401

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXX-- -------- -------- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.

Errors   :     0
Warnings :     1
```

NOTES:

© 1994 Microchip Technology Inc.

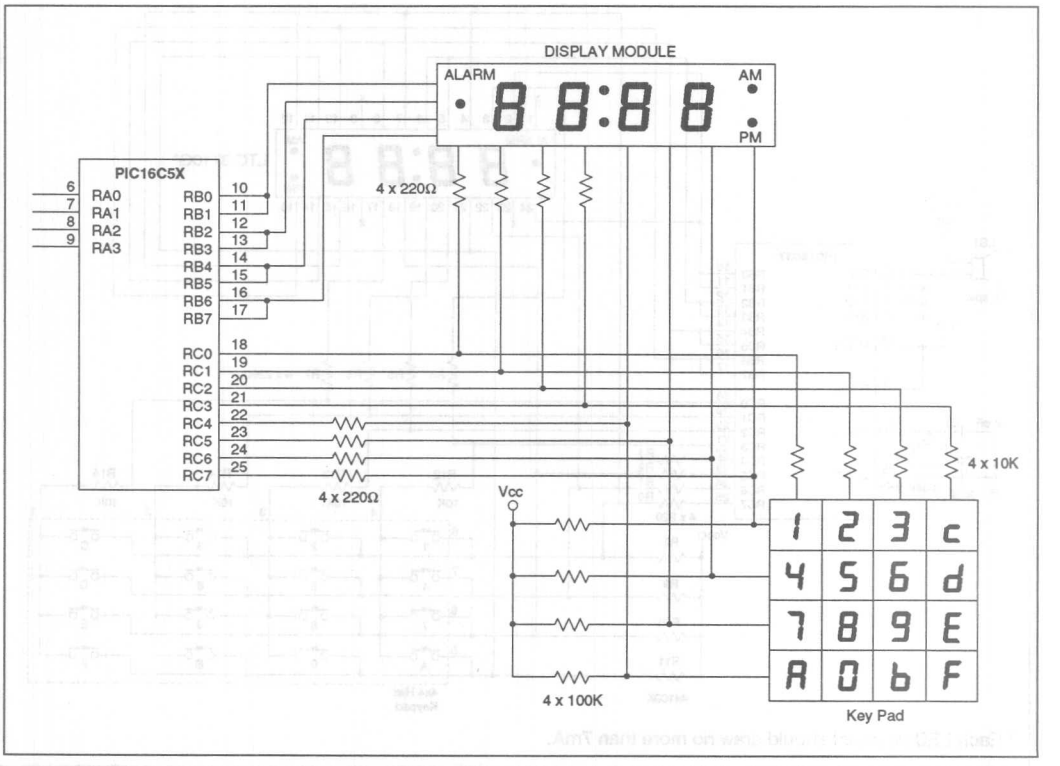
Multiplexing LED Drive and a 4x4 Keypad Sampling

INTRODUCTION

Many applications require driving LEDs along with an interface to a keypad. Implementing such designs usually involves using up significant amounts of the processors I/O lines. This application note describes a method which uses only 16 I/O pins of a PIC16C5X microcontroller to sample a 4x4 keypad matrix, and directly drive four 7 segment LEDs (see Figure 1). Direct drive of the LEDs is possible, because of the high sink and source capabilities of the PIC16C5X microcontroller, thus eliminating the use of external drive transistor, and resulting in reduced cost and complexity of the overall circuit.

Typically applications having LEDs and keypads also keep track of real time, in order to synchronize certain key events. An Industrial Clock/Timer example has been used in this application note as a demonstration of this technique. The software overhead to keep track of real time is minimal and the user can modify the code to significantly expand the functionality of this circuit.

FIGURE 1 - PIC16C5X INTERFACE TO A SEGMENT DISPLAY AND 4X4 KEYPAD



© 1994 Microchip Technology Incorporated

Multiplexing LED Drive and a 4x4 Keypad Sampling

PART A: 4X4 KEY MATRIX SAMPLING

Implementation

The 4x4 Key Matrix is connected to port C of the PIC16C5X (Figure 2a). The four columns are connected to RC0-RC3 and the four rows are connected to RC4-RC7. Each digit is refreshed every 20 ms. with a 5 ms pulse. The keypad is sampled every 20 ms with four 3 µs pulses (Figure 3).

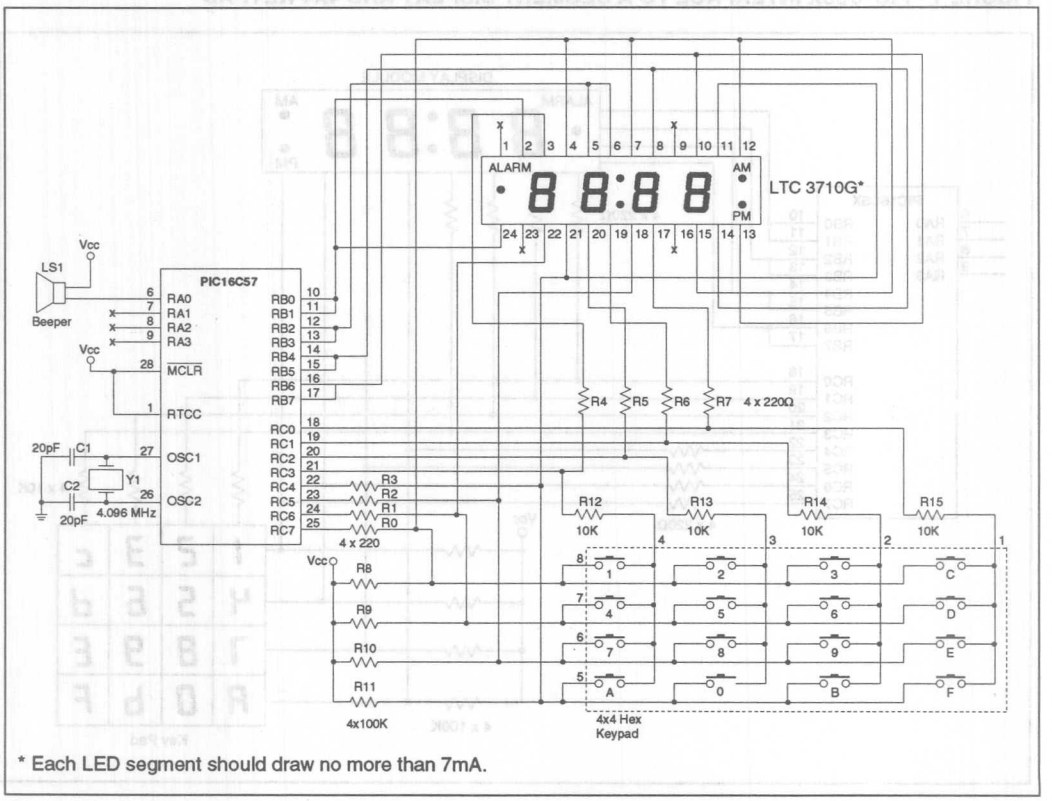
The keypad sampling is as follows:

1. The columns are connected to output pins, and the rows are connected to input pins.
2. Each column is sequentially driven to a low voltage while at the same instance the four rows are sampled. Since the rows are all held high with pull-up resistors, all four inputs will normally be high. If a key is pressed in a column which is at a low level, that low level will be conducted to the input pin through the closed key and the corresponding row will be sensed as a low.
3. Before a new column is brought low, care should be taken to discharge the input pins (see code section for details).
4. A 50 ms key debounce technique has been implemented in the software, in order to eliminate multiple key strokes.

Notes:

1. Resistors R8-R11 and R12-R14 have been selected such that their ratio is 1:10. This will insure a 0.5 Volt level at the input, when a key is pressed. Also R8-R14 should have a value such that their current contribution to the LEDs segments is negligible.
2. In circuits where there is substantial interference between the key matrix and the LED drive circuit, the alternative circuit (Figure 2b) should be utilized. Diodes in the path of all pins connected to the keypad insure that there is minimal interference from the keypad, when it is not being sampled.

FIGURE 2A - PIC16C5X INDUSTRIAL CLOCK/TIMER SCHEMATIC



* Each LED segment should draw no more than 7mA.

© 1994 Microchip Technology Incorporated

PART B: INDUSTRIAL CLOCK/TIMER

Clock Selection

The 4.096 MHz crystal oscillator is the time base. The PIC16C5X internally divides the clock by 4 to give an internal clock of 1.024 MHz. This clock is further divided by 32 (by the prescaler in the OPTIONS register) to give a clock of 32 KHz which is used to increment the RTCC in the PIC16C5X. If the RTCC is initialized to 96, it would overflow to 0 in 5 ms.

(256-96) x (1/32000) = 5.000 ms

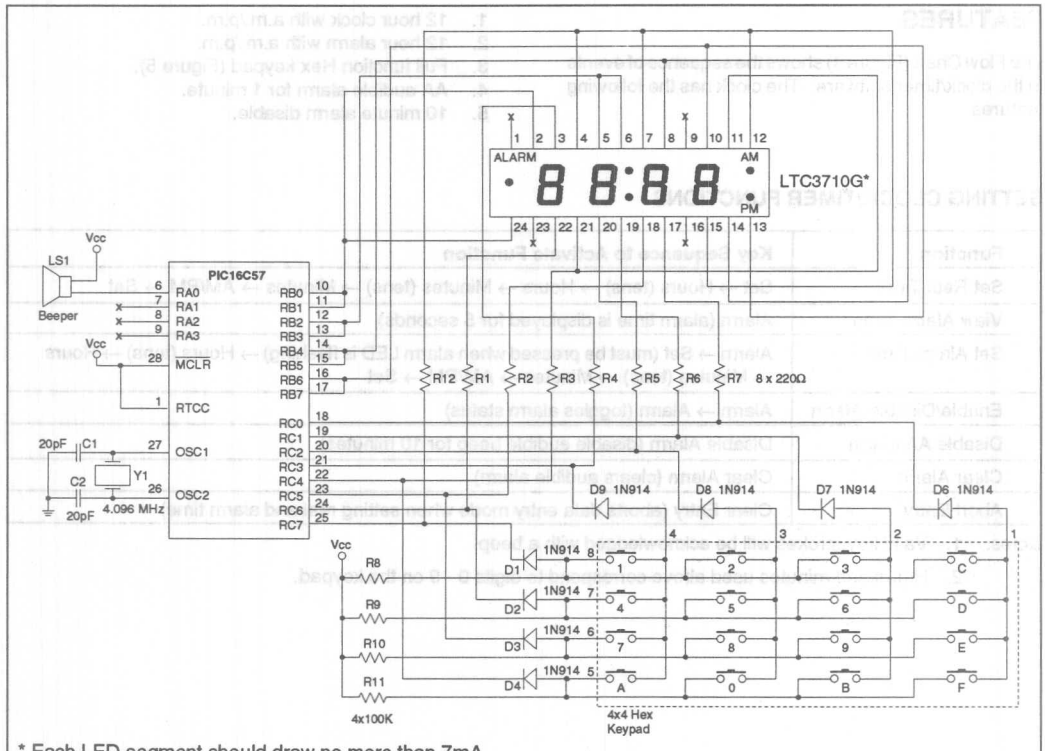
This 5 ms is used to count the seconds, minutes and hours in the clock/timer. It is also used as a time base to update the display digits and sample the keyboard. The clock speed being 4.096 MHz, each instruction will execute in 1 µs. Therefore in 5 ms, approximately 5000 instructions can be executed. This gives sufficient time to execute a large section of code and not miss the overflow in the RTCC.

Using a 3.579545 MHz color burst crystal oscillator as a time base

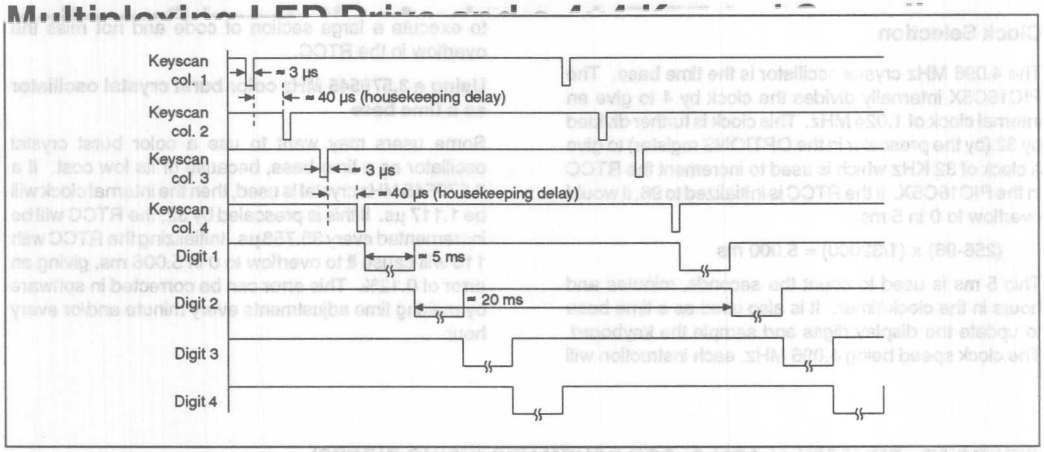
Some users may want to use a color burst crystal oscillator as a time base, because of its low cost. If a 3.579545 MHz crystal is used, then the internal clock will be 1.117 µs. If this is prescaled by 32, the RTCC will be incremented every 35.758 µs. Initializing the RTCC with 116 will cause it to overflow to 0 in 5.006 ms. giving an error of 0.12%. This error can be corrected in software by making time adjustments every minute and/or every hour.

2

FIGURE 2B - PIC16C5X ALARM CLOCK SCHEMATIC (USING DIODES)



* Each LED segment should draw no more than 7mA.



FEATURES

The Flow Chart (Figure 4) shows the sequence of events in the clock/timer software. The clock has the following features:

1. 12 hour clock with a.m./p.m.
2. 12 hour alarm with a.m./p.m.
3. Full function Hex keypad (Figure 5).
4. AA audible alarm for 1 minute.
5. 10 minute alarm disable.

SETTING CLOCK/TIMER FUNCTIONS

Function	Key Sequence to Activate Function
Set Real Time	Set → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
View Alarm Time	Alarm (alarm time is displayed for 5 seconds)
Set Alarm Time	Alarm → Set (must be pressed when alarm LED is flashing) → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
Enable/Disable Alarm	Alarm → Alarm (toggles alarm status)
Disable AA alarm	Disable Alarm (disable audible beep for 10 minutes)
Clear Alarm	Clear Alarm (clears audible alarm)
Abort Entry	Clear Entry (aborts data entry mode when setting real and alarm time)

Notes: 1. Valid key strokes will be acknowledged with a beep.

2. Hours and minutes used above correspond to digits 0 - 9 on the keypad.

© 1994 Microchip Technology Incorporated

FIGURE 4 - TIMER/CLOCK FLOW CHART

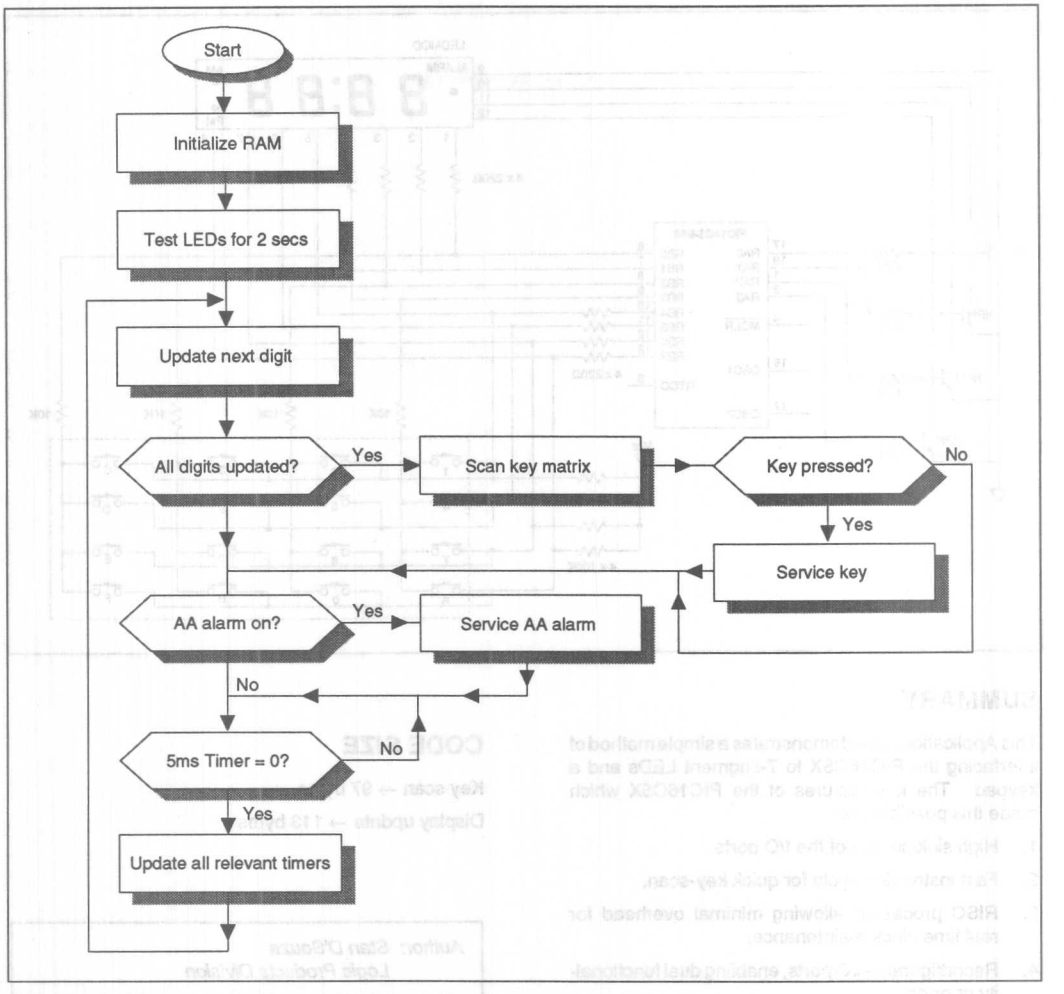
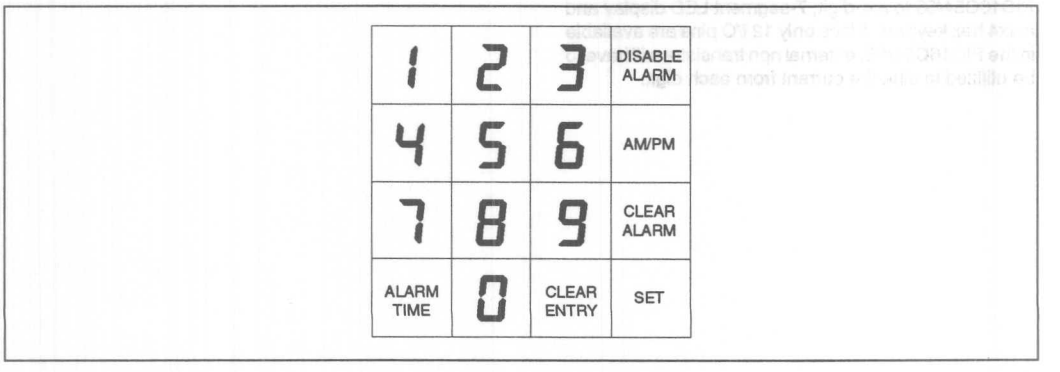
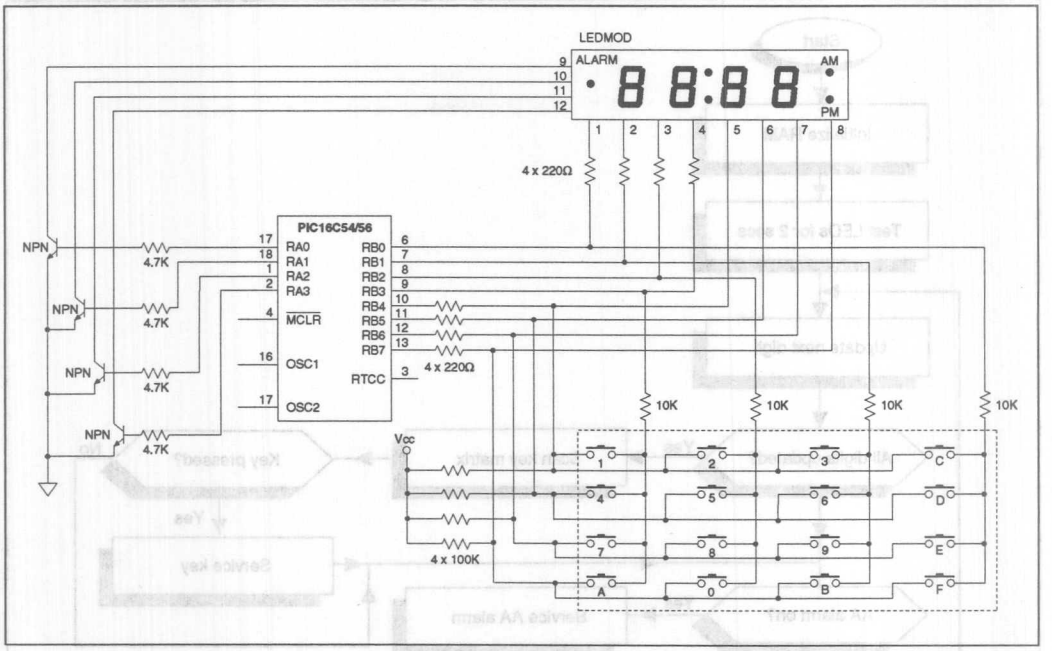


FIGURE 5 - KEYPAD



© 1994 Microchip Technology Incorporated

FIGURE 6 - INTERFACE TO PIC16C54/56



SUMMARY

This Application Note demonstrates a simple method of interfacing the PIC16C5X to 7-segment LEDs and a keypad. The key features of the PIC16C5X which made this possible are:

1. High sink/source of the I/O ports.
2. Fast instruction cycle for quick key-scan.
3. RISC processor allowing minimal overhead for real time clock maintenance.
4. Reconfigurable I/O ports, enabling dual functionality of ports.

Figure 6 depicts a block diagram connecting a PIC16C54/56 to a 4-digit, 7-segment LED display and a 4x4 hex keypad. Since only 12 I/O pins are available in the PIC16C54/56, external npn transistor will have to be utilized to sink the current from each digit.

CODE SIZE

Key scan → 97 bytes

Display update → 113 bytes

Author: Stan D'Souza
Logic Products Division

© 1994 Microchip Technology Incorporated

APPENDIX A: CODE LISTING

```
MPASM 1.00 Released       CLK.ASM   7-15-1994  13:15:10                 PAGE  1
Alarm Clock

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001                    TITLE       "Alarm Clock"
                     0002               LIST  P = 16C57,f=inhx8m
                     0003 ;
                     0004 ;Define Equates:
                     0005 ;
07FF                 0006               PIC57   EQU     7FFH
                     0007
;*****************************************************************
                     0008 ;External Ossc. used = 4.096Mhz. Prescaler of 32 used, which gives a
                     0009 ;31.25 microSec increment of the RTCC. If RTCC is intially loaded with
96,
                     0010 ;it would overflow to 0 in 5.000 milliSecs. Giving a 0.00% error.
0060                 0011               MSEC5   EQU     D'96'
                     0012
;*****************************************************************
0000                 0013               C       EQU     0
0000                 0014               BEP     EQU     0
0000                 0015               RTATS   EQU     0
0001                 0016               DC      EQU     1
0001                 0017               HR10    EQU     1
0002                 0018               Z       EQU     2
0002                 0019               HR      EQU     2
0003                 0020               MIN10   EQU     3
0004                 0021               MIN     EQU     4
0004                 0022               FLASH   EQU     4
0005                 0023               PA0     EQU     5
0005                 0024               KEY_BEEP EQU    5
0005                 0025               AMPM    EQU     5
0006                 0026               PA1     EQU     6
0000                 0027               F0      EQU     0
0006                 0028               KEY_HIT EQU     6
0006                 0029               ALED    EQU     6
0007                 0030               AM_PM   EQU     7
0003                 0031               COLON   EQU     3
0002                 0032               ALRMLED EQU     2
0007                 0033               SERVICED EQU    7
0000                 0034               ALONOF  EQU     0
0001                 0035               INAL    EQU     1
0002                 0036               SILNC   EQU     2
0003                 0037               INAA    EQU     3
0005                 0038               INKEYBEP EQU    5
                     0039 ;
                     0040 ;DEFINE RAM LOCATIONS:
0001                 0041               RTCC    EQU     1
0002                 0042               PC      EQU     2
0003                 0043               STATUS  EQU     3
0004                 0044               FSR     EQU     4
0005                 0045               PORT_A  EQU     5
0006                 0046               PORT_B  EQU     6
0007                 0047               PORT_C  EQU     7
                     0048 ;DEFINE REAL TIME MODE REGS (RTM)
0008                 0049               MSTMR   EQU     8       ;MILLI SEC. TIMER
0009                 0050               STMR    EQU     9       ;SEC. TIMER
                     0051 ;*************************************************
                     0052 ;DO NOT CHANGE RELATIVE POSITION OF NEXT 6 BYTES
000A                 0053               MTMR    EQU     0A      ;MIN. TIMER
000B                 0054               HTMR    EQU     0B      ;HOUR TIMER
                     0055 ;DEFINE ALARM TIME MODE REGS (ATM)
000C                 0056               MALARM  EQU     0C      ;MIN. ALARM
000D                 0057               HALARM  EQU     0D      ;HOUR ALARM
                     0058 ;DEFINE DATA ENTRY MODE REGS (DEM)
```

© 1994 Microchip Technology Incorporated

```
000E                0059          MENTRY  EQU     0E      ;MIN. ENTRY
000F                0060          HENTRY  EQU     0F      ;HOUR ENTRY
                    0061 ;*****************************************************
                    0063 ;
                    0064 ;DEFINE FLAG REG AND FUNCTION:
0010                0065          FLAG    EQU     10
                    0066 ;        BIT # 7|6|5|4|3|2|1|0|
                    0067 ;--------|-|-|-|-|-|-|-|-|
                    0068 ;                X|X|X|X|X|X|0|0| -> REAL TIME MODE (RTM)
                    0069 ;                X|X|X|X|X|X|0|1| -> ALARM TIME MODE(ATM)
                    0070 ;                X|X|X|X|X|X|1|0| -> DATA ENTRY M0DE(DEM)
                    0071 ;                X|X|X|X|X|X|1|1| -> TEST MODE (TM)
                    0072 ;                X|X|X|X|X|Y|X|X| -> ALRMLED ON/OFF
                    0073 ;                X|X|X|X|Y|X|X|X| -> COLON LED ON/OFF
                    0074 ;                X|X|X|Y|X|X|X|X| -> FLASH DISPLAY
                    0075 ;                X|X|Y|X|X|X|X|X| -> KEY_BEEP
                    0076 ;                X|Y|X|X|X|X|X|X| -> KEY_HIT (0/1)
                    0077 ;                Y|X|X|X|X|X|X|X| -> SERVICED
                    0078 ; X = DEFINED ELSEWHERE IN TABLE
                    0079 ; Y = DEFINED AS SHOWN (0/1)
                    0080 ;
0011                0081          TEMP    EQU     11
0012                0082          DIGIT   EQU     12
0013                0083          NEW_KEY EQU     13
0014                0084          KEY_NIBL EQU    14
0015                0085          DEBOUNCE EQU    15
0016                0086          MIN_SEC EQU     16      ;MIN/SECONDS TIMER
0017                0087          ENTFLG  EQU     17
                    0088 ;flag dedicated to the key entry mode
                    0089 ;        BIT # 7|6|5|4|3|2|1|0|
                    0090 ;--------|-|-|-|-|-|-|-|-|
                    0091 ;                X|X|X|X|X|X|X|Y| -> REAL/ALARM TIME STATUS
                    0092 ;                X|X|X|X|X|X|Y|X| -> HR10 DONE
                    0093 ;                X|X|X|X|X|Y|X|X| -> HR DONE
                    0094 ;                X|X|X|X|Y|X|X|X| -> MIN10 DONE
                    0095 ;                X|X|X|Y|X|X|X|X| -> MIN DONE
                    0096 ;                X|X|Y|X|X|X|X|X| -> INKEYBEP
                    0097 ;                X|Y|X|Y|X|X|X|X| -> NOT USED
                    0098 ;                Y|X|X|X|X|X|X|X| -> NOT USED
                    0099 ;
                    0100 ;
0018                0101          ALFLAG  EQU     18
                    0102 ;flag dedicated to the alarm
                    0103 ;        BIT # 7|6|5|4|3|2|1|0|
                    0104 ;--------|-|-|-|-|-|-|-|-|
                    0105 ;                X|X|X|X|X|X|X|Y| -> ALONOF
                    0106 ;                X|X|X|X|X|X|Y|X| -> INAL
                    0107 ;                X|X|X|X|X|Y|X|X| -> SILNC
                    0108 ;                X|X|X|X|Y|X|X|X| -> INAA
                    0109 ;                X|X|X|Y|X|X|X|X| -> NOT USED
                    0110 ;                X|X|Y|X|X|X|X|X| -> NOT USED
                    0111 ;                X|Y|X|Y|X|X|X|X| -> NOT USED
                    0112 ;                Y|X|X|X|X|X|X|X| -> NOT USED
                    0113 ;
0019                0114          AAFLAG  EQU     19
                    0115 ;flag dedicated to the AA alarm
001A                0116          AATMR   EQU     1A
                    0117 ;
                    0118 ;Port pin definitions:
                    0119 ;
                    0120 ;PORT_A:
                    0121 ;        BIT 0   -> BEEPER (ACTIVE LOW) OUTPUT
                    0122 ;        BIT 1-3 -> unused I/O
                    0123 ;
                    0124 ;PORT_B: ALL OUTPUTS
                    0125 ;        BIT 0&4 -> MSB DIGIT COMMON CATHODE & ALARM
                    0126 ;        BIT 1&5 -> 2ND DIGIT COMMOM CATHODE & COLON
                    0127 ;        BIT 2&6 -> 3RD DIGIT COMMON CATHODE & PM
                    0128 ;        BIT 3&7 -> LSB DIGIT COMMON CATHODE & AM
```


© 1994 Microchip Technology Incorporated

```
                    0129 ;
                    0130 ;PORT_C:
                    0131 ;IN DISPLAY MODE ALL SEG/ANNN SET AS OUTPUTS
                    0132 ;IN KEY SCAN MODE COLS ARE OUTPUTS ROWS ARE INPUTS
                    0133 ;       BIT 0   -> SEGMENT A & COL 4
                    0134 ;       BIT 1   -> SEGMENT B & COL 3
                    0135 ;       BIT 2   -> SEGMENT C & COL 2
                    0136 ;       BIT 3   -> SEGMENT D & COL 1
                    0137 ;       BIT 4   -> SEGMENT E & ROW 4
                    0138 ;       BIT 5   -> SEGMENT F & ROW 3
                    0139 ;       BIT 6   -> SEGMENT G & ROW 2
                    0140 ;       BIT 7   -> CA OF ALL ANNUNCIATORS & ROW 1
                    0141 ;
                    0142 ;
                    0144 ;
                    0145         ORG     0
                    0146 START
0000 0AFC           0147         GOTO    INIT_CLK        ;INITIALIZE CLOCK
                    0148 ;THIS ROUTINE RUNS A TEST ON THE LEDS.
                    0149 ;ALL THE RELEVENT LEDS ARE LIT UP FOR 2 SECS.
                    0150 ;
                    0151 TEST_HARDWARE
0001 0C02           0152         MOVLW   d'02'           ;DISPLAY FOR 2 SECS
0002 0036           0153         MOVWF   MIN_SEC         ;       /
                    0154 ;
                    0155 ;
                    0156 TEST_LOOP
0003 0216           0157         MOVF    MIN_SEC,W       ;GET MIN/SEC
0004 0643           0158         BTFSC   STATUS,Z        ;NOT 0 THEN SKIP
0005 0A0B           0159         GOTO    NORM_TIME       ;ELSE NORMAL TIME
0006 0925           0160         CALL    UPDATE_DISPLAY  ;UPDATE DISPLAY
0007 05A3           0161         BSF     STATUS,PA0      ;GOTO PAGE 1
0008 0900           0162         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE
0009 04A3           0163         BCF     STATUS,PA0      ;RESET PAGE MARKER
000A 0A03           0164         GOTO    TEST_LOOP       ;LOOP BACK
                    0165 NORM_TIME
000B 0410           0166         BCF     FLAG,0          ;PUT IN REAL TIME
000C 0430           0167         BCF     FLAG,1
                    0168 TIME_LOOP
000D 0925           0169         CALL    UPDATE_DISPLAY
000E 05C3           0170         BSF     STATUS,PA1      ;GOTO PAGE 2
000F 0900           0171         CALL    SERVICE_KEYS
0010 05A3           0172         BSF     STATUS,PA0      ;GOTO PAGE 3
0011 0900           0173         CALL    SOUND_AA        ;CHECK ALARM
0012 04C3           0174         BCF     STATUS,PA1      ;GOTO PAGE 1
0013 0900           0175         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE TIMERS
0014 04A3           0176         BCF     STATUS,PA0      ;RESET PAGE MARKER
0015 04C3           0177         BCF     STATUS,PA1      ;       /
0016 0210           0178         MOVF    FLAG,W          ;SEE IF IN ATM
0017 0E03           0179         ANDLW   B'00000011'     ;       /
0018 0F01           0180         XORLW   B'00000001'     ;      /
0019 0643           0181         BTFSC   STATUS,Z        ;SKIP IF NOT
001A 091C           0182         CALL    RESET_ATM
001B 0A0D           0183         GOTO    TIME_LOOP
                    0184 ;
                    0185 RESET_ATM
001C 0216           0186         MOVF    MIN_SEC,W       ;GET MIN/SEC
001D 0E0F           0187         ANDLW   B'00001111'     ;       /
001E 0743           0188         BTFSS   STATUS,Z        ;Z THEN SKIP
001F 0800           0189         RETLW   0               ;ELSE RETURN
0020 0410           0190         BCF     FLAG,0          ;SET TO RTM
0021 0450           0191         BCF     FLAG,ALRMLED    ;CLEAR LED
0022 0618           0192         BTFSC   ALFLAG,ALONOF   ;TEST STAT
0023 0550           0193         BSF     FLAG,ALRMLED    ;SET LED
0024 0800           0194         RETLW   0               ;RETURN
                    0196 ;
                    0197 ;
                    0198 UPDATE_DISPLAY
0025 0C00           0199         MOVLW   B'00000000'     ;CLEAR SEG DRIVE
```

© 1994 Microchip Technology Incorporated

```
0026 0027           0200          MOVWF   PORT_C          ;       /
0027 0C3F           0201          MOVLW   B'00111111'     ;SEE IF LAST DIGIT
0028 0186           0202          XORWF   PORT_B,0        ;       /
0029 0643           0203          BTFSC   STATUS,Z        ;NO THEN SKIP
002A 0A6F           0204          GOTO    SCAN_KP         ;ELSE SCAN KEYPAD
                    0205 UP_DSP_1
                    0206 ;SELECT DIGIT TO BE DISPLAYED
002B 0246           0207          COMF    PORT_B,0        ;GET COMPL. PORT B IN W
002C 0643           0208          BTFSC   STATUS,Z        ;NO DIGIT SELECTED?
002D 0CC0           0209          MOVLW   B'11000000'     ;THEN SELECT DEFAULT
002E 0031           0210          MOVWF   TEMP            ;SAVE IN TEMP
002F 0271           0211          COMF    TEMP            ;COMPLEMENT VALUE
0030 0503           0212          BSF     STATUS,C        ;SET CARRY
0031 0371           0213          RLF     TEMP            ;SHIFT LEFT
0032 0703           0214          BTFSS   STATUS,C        ;IF C=1 THEN SKIP
0033 0371           0215          RLF     TEMP            ;ELSE 3 TIMES...
0034 0371           0216          RLF     TEMP            ;THRU CARRY
0035 0211           0217          MOVF    TEMP,0          ;GET IN W
0036 0026           0218          MOVWF   PORT_B          ;OUTPUT TO PORT
                    0219 ;NOW THAT DIGIT IS SELECTED, SELECT SEG VALUES FOR THAT DIGIT
                    0220 ;FIRST FIND MODE OF OPERATION.
0037 0C0A           0221          MOVLW   MTMR            ;LOAD FSR WITH MTMR
0038 0024           0222          MOVWF   FSR             ;       /
0039 0210           0223          MOVF    FLAG,0          ;GET FLAG IN W
003A 0E03           0224          ANDLW   B'00000011'     ;MASK OTHER BITS
003B 0031           0225          MOVWF   TEMP            ;SAVE IN TEMP
003C 0F03           0226          XORLW   B'00000011'     ;IN TEST MODE
003D 0643           0227          BTFSC   STATUS,Z        ;NO THEN SKIP
003E 0A4B           0228          GOTO    DO_TM           ;ELSE TEST MODE
003F 0403           0229          BCF     STATUS,C        ;CLEAR CARRY
0040 0371           0230          RLF     TEMP            ;LEFT SHIFT TEMP
0041 0211           0231          MOVF    TEMP,0          ;GET IN W
0042 01E4           0232          ADDWF   FSR             ;CHANGE INDIRECT POINTER
0043 0954           0233          CALL    GET_7_SEG       ;GET 7 SEG DATA IN W
0044 0032           0234          MOVWF   DIGIT           ;SAVE IN DIGIT LOC.
0045 09D1           0235          CALL    MASK_ANNC       ;MASK ANNC TO DIGIT
0046 0690           0236          BTFSC   FLAG,FLASH      ;NO FLASH THEN SKIP
0047 094E           0237          CALL    CHK_HALF_SEC    ;ELSE CHK. IF ON
0048 0212           0238          MOVF    DIGIT,0         ;GET BACK DIGIT
0049 0027           0239          MOVWF   PORT_C          ;OUTPUT TO PORT
004A 0800           0240          RETLW   0               ;RETURN
                    0241 ;
                    0242 DO_TM
004B 0CFF           0243          MOVLW   B'11111111'     ;LIGHT ALL SEGMENTS
004C 0027           0244          MOVWF   PORT_C          ;       /
004D 0800           0245          RETLW   0               ;RETURN FROM UPDATE DISPLAY
                    0246 ;
                    0247 CHK_HALF_SEC
004E 0770           0248          BTFSS   FLAG,COLON      ;IF COLON ON THEN DO
004F 0A51           0249          GOTO    BLANK_DSP       ;ELSE BLANK DISPLAY
0050 0800           0250          RETLW   0
                    0251 BLANK_DSP
0051 0C00           0252          MOVLW   B'00000000'     ;MAKE PORT C LOW
0052 0032           0253          MOVWF   DIGIT
0053 0800           0254          RETLW   0
                    0255 ;
                    0257 ;
                    0258 ;ON ENTRY FSR POINTS TO THE REAL TIME MODE'S MINUTES REGISTER.
                    0259 ;ON RETURN FSR POINTS TO THE TIMER REGISTER TO BE DISPLAYED.
                    0260 ;W REG. CONTAINS THE DECODED 7 SEG. INFO OF THE DIGIT
                    0261 ;TO BE DISPLAYED
                    0262 ;
                    0263 GET_7_SEG
0054 0246           0264          COMF    PORT_B,0        ;COMPLEMENT B -> W
0055 0EF0           0265          ANDLW   B'11110000'     ;MASK LO NIBBLE
0056 0643           0266          BTFSC   STATUS,Z        ;NZ THEN SKIP
0057 02A4           0267          INCF    FSR             ;INC POINTER
0058 0200           0268          MOVF    F0,0            ;MOVE INDIRECT TO W
0059 0031           0269          MOVWF   TEMP            ;GET INTO TEMP
```


© 1994 Microchip Technology Incorporated

```
005A 0246          0270         COMF    PORT_B,0          ;COMPL.B -> W
005B 0EF0          0271         ANDLW   B'11110000'       ;MASK LO NIBBLE
005C 0643          0272         BTFSC   STATUS,Z          ;IF D1/2 THEN
005D 04F1          0273         BCF     TEMP,AM_PM        ;CLEAR AM/PM BIT
005E 0246          0274         COMF    PORT_B,0          ;GET PORT B AGAIN
005F 0ECC          0275         ANDLW   B'11001100'       ;SEE IF D2 OR D4
0060 0643          0276         BTFSC   STATUS,Z          ;YES THEN SKIP
0061 03B1          0277         SWAPF   TEMP              ;SWAP TEMP
0062 0C0F          0278         MOVLW   B'00001111'       ;MASK HI NIBBLE
0063 0151          0279         ANDWF   TEMP,0
0064 01E2          0280         ADDWF   PC                ;ADD TO PC
0065 083F          0281         RETLW   B'00111111'       ;CODE FOR 0
0066 0806          0282         RETLW   B'00000110'       ;CODE FOR 1
0067 085B          0283         RETLW   B'01011011'       ;CODE FOR 2
0068 084F          0284         RETLW   B'01001111'       ;CODE FOR 3
0069 0866          0285         RETLW   B'01100110'       ;CODE FOR 4
006A 086D          0286         RETLW   B'01101101'       ;CODE FOR 5
006B 087D          0287         RETLW   B'01111101'       ;CODE FOR 6
006C 0807          0288         RETLW   B'00000111'       ;CODE FOR 7
006D 087F          0289         RETLW   B'01111111'       ;CODE FOR 8
006E 0867          0290         RETLW   B'01100111'       ;CODE FOR 9
                   0291 ;
                   0292 ;This routine scans the 4x4 hex key pad for a key hit.
                   0293 ;If key is pressed, KEY_HIT flag is set and the value of
                   0294 ;the hex key is returned in reg NEW_KEY
                   0295 ;If no key is detected, then a 0xff value is returned in
                   0296 ;register NEW_KEY and the flag KEY_HIT is reset.
                   0297 ;
                   0298 SCAN_KP
006F 06D0          0299         BTFSC   FLAG,KEY_HIT      ;KEY UNDER SERVICE?
0070 0A2B          0300         GOTO    UP_DSP_1          ;YES SKIP ROUTINE
0071 0CFF          0301         MOVLW   B'11111111'       ;SET DIGIT SINKS ...
0072 0026          0302         MOVWF   PORT_B            ;TO HIGH
0073 0CF7          0303         MOVLW   B'11110111'       ;SET KEY COL LOW
0074 0031          0304         MOVWF   TEMP              ;SAVE IN TEMP
                   0305 SKP1
0075 0C00          0306         MOVLW   B'00000000'       ;SET PORT C AS OUTPUTS
0076 0007          0307         TRIS    PORT_C            ;       /
0077 0211          0308         MOVF    TEMP,W
0078 0E0F          0309         ANDLW   B'00001111'       ;DISCHARGE PINS
0079 0027          0310         MOVWF   PORT_C            ;       /
007A 0CF0          0311         MOVLW   B'11110000'       ;SET AS I/O
007B 0007          0312         TRIS    PORT_C            ;    /
007C 0211          0313         MOVF    TEMP,W            ;GET OLD VALUE
007D 0027          0314         MOVWF   PORT_C            ;OUTPUT TO PORT
007E 0207          0315         MOVF    PORT_C,W          ;INPUT PORT VALUE
007F 0EF0          0316         ANDLW   B'11110000'       ;MASK LO BYTE
0080 0FF0          0317         XORLW   B'11110000'       ;SEE IF KEY HIT
0081 0743          0318         BTFSS   STATUS,Z          ;NO KEY THEN SKIP
0082 0A8D          0319         GOTO    DET_KEY           ;LOAD KEY VALUE
                   0320 SKP3
0083 0503          0321         BSF     STATUS,C          ;SET CARRY
0084 0331          0322         RRF     TEMP              ;MAKE NEXT COL. LOW
0085 0603          0323         BTFSC   STATUS,C          ;ALL DONE THEN SKIP
0086 0A75          0324         GOTO    SKP1
0087 0073          0325         CLRF    NEW_KEY           ;SET NEW_KEY = FF
0088 00F3          0326         DECF    NEW_KEY           ;       /
                   0327 SKP2
0089 0067          0328         CLRF    PORT_C            ;SETPORT C AS ...
008A 0C00          0329         MOVLW   B'00000000'       ;OUTPUTS
008B 0007          0330         TRIS    PORT_C            ;       /
008C 0A2B          0331         GOTO    UP_DSP_1          ;RETURN
                   0332 DET_KEY
                   0333 ;key is detected
008D 0293          0334         INCF    NEW_KEY,W         ;CHK IF KEY ...
008E 0743          0335         BTFSS   STATUS,Z          ;WAS RELEASED
008F 0A89          0336         GOTO    SKP2              ;NO THEN RETURN
0090 0207          0337         MOVF    PORT_C,W          ;GET RAW KEY...
0091 0D0F          0338         IORLW   B'00001111'       ;VALUE.
```

© 1994 Microchip Technology Incorporated

```
0092 0151           0339          ANDWF   TEMP,W          ;   /
0093 0033           0340          MOVWF   NEW_KEY         ;SAVE IN NEW_KEY
0094 0998           0341          CALL    GET_KEY_VAL     ;GET ACTUAL KEY ...
0095 0033           0342          MOVWF   NEW_KEY         ;VALUE
0096 05D0           0343          BSF     FLAG,KEY_HIT    ;SET KEY HIT FLAG
0097 0A89           0344          GOTO    SKP2            ;RETURN
                    0345 ;
                    0347 ;This routine decodes the hex value from the "raw" data got
                    0348 ;from scanning the rows and cols.
                    0349 ;   actual key value          raw hex value
                    0350 ;          ONE             EQU     77
                    0351 ;          TWO             EQU     7B
                    0352 ;          THREE           EQU     7D
                    0353 ;          C               EQU     7E
                    0354 ;          FOUR            EQU     0B7
                    0355 ;          FIVE            EQU     0BB
                    0356 ;          SIX             EQU     0BD
                    0357 ;          D               EQU     0BE
                    0358 ;          SEVEN           EQU     0D7
                    0359 ;          EIGHT           EQU     0DB
                    0360 ;          NINE            EQU     0DD
                    0361 ;          E               EQU     0DE
                    0362 ;          A               EQU     0E7
                    0363 ;          ZERO            EQU     0EB
                    0364 ;          B               EQU     0ED
                    0365 ;          F               EQU     0EE
                    0366 ;
                    0367 ;
                    0368 GET_KEY_VAL
0098 0E0F           0369          ANDLW   B'00001111'     ;GET LO NIBBLE
0099 0034           0370          MOVWF   KEY_NIBL        ;SAVE
009A 0C04           0371          MOVLW   4               ;SET COUNT TO 4
009B 0031           0372          MOVWF   TEMP            ;       /
                    0373 GKV1
009C 0503           0374          BSF     STATUS,C        ;SET CARRY
009D 0334           0375          RRF     KEY_NIBL        ;ROTATE NIBBLE
009E 0703           0376          BTFSS   STATUS,C        ;SKIP IF NOT Z
009F 0AA5           0377          GOTO    GET_HI_KEY      ;GOTO NEXT PART
00A0 02F1           0378          DECFSZ  TEMP            ;DEC COUNT
00A1 0A9C           0379          GOTO    GKV1            ;LOOP
                    0380 GO_RESET
00A2 05A3           0381          BSF     STATUS,PA0      ;SET MSB
00A3 05C3           0382          BSF     STATUS,PA1      ;       /
00A4 0BFF           0383          GOTO    SYS_RESET       ;ELSE BIG ERROR
                    0384 GET_HI_KEY
00A5 00F1           0385          DECF    TEMP            ;REDUCE BY 1
00A6 0393           0386          SWAPF   NEW_KEY,W       ;GET HI NIBBLE
00A7 0E0F           0387          ANDLW   B'00001111'     ;       /
00A8 0034           0388          MOVWF   KEY_NIBL        ;SAVE
00A9 0211           0389          MOVF    TEMP,W          ;GET OFFSET TO TBL
00AA 01E2           0390          ADDWF   PC              ;LOAD IN PC
00AB 0AAF           0391          GOTO    GET147A         ;JUMP TO NEXT PART
00AC 0AB8           0392          GOTO    GET2580         ;       /
00AD 0ABA           0393          GOTO    GET369B         ;      /
00AE 0ABC           0394          GOTO    GETCDEF         ;     /
                    0395 ;
                    0396 GET147A
00AF 0C04           0397          MOVLW   4               ;SET COUNT TO 4
                    0398 GETCOM
00B0 0031           0399          MOVWF   TEMP            ;
                    0400 GETCOM1
00B1 0503           0401          BSF     STATUS,C        ;SET CARRY
00B2 0334           0402          RRF     KEY_NIBL        ;ROTATE RIGHT
00B3 0703           0403          BTFSS   STATUS,C        ;CHECK IF DONE
00B4 0ABE           0404          GOTO    KEY_TBL         ;JUMP TO TABLE
00B5 02F1           0405          DECFSZ  TEMP            ;DEC COUNT
00B6 0AB1           0406          GOTO    GETCOM1         ;LOOP
00B7 0AA2           0407          GOTO    GO_RESET        ;ELSE ERROR
                    0408 ;
```


© 1994 Microchip Technology Incorporated

```
                    0409 GET2580
00B8 0C08           0410         MOVLW   8              ;SET COUNT TO 8
00B9 0AB0           0411         GOTO    GETCOM
                    0412 ;
                    0413 GET369B
00BA 0C0C           0414         MOVLW   D'12'          ;SET COUNT TO 12
00BB 0AB0           0415         GOTO    GETCOM
                    0416 ;
                    0417 GETCDEF
00BC 0C10           0418         MOVLW   D'16'          ;SET COUNT TO 16
00BD 0AB0           0419         GOTO    GETCOM
                    0421 ;
                    0422 KEY_TBL
00BE 00F1           0423         DECF    TEMP           ;REDUCE BY 1
00BF 0211           0424         MOVF    TEMP,W         ;GET IN W
00C0 01E2           0425         ADDWF   PC             ;JUMP TO TABLE
00C1 0801           0426         RETLW   1              ;KEY 1
00C2 0804           0427         RETLW   4              ;KEY 4
00C3 0807           0428         RETLW   7              ;KEY 7
00C4 080A           0429         RETLW   0A             ;KEY A
00C5 0802           0430         RETLW   2              ;KEY 2
00C6 0805           0431         RETLW   5              ;KEY 5
00C7 0808           0432         RETLW   8              ;KEY 8
00C8 0800           0433         RETLW   0              ;KEY 0
00C9 0803           0434         RETLW   3              ;KEY 3
00CA 0806           0435         RETLW   6              ;KEY 6
00CB 0809           0436         RETLW   9              ;KEY 9
00CC 080B           0437         RETLW   0B             ;KEY B
00CD 080C           0438         RETLW   0C             ;KEY C
00CE 080D           0439         RETLW   0D             ;KEY D
00CF 080E           0440         RETLW   0E             ;KEY E
00D0 080F           0441         RETLW   0F             ;KEY F
                    0442 ;
                    0444 ;
                    0445 MASK_ANNC
00D1 0CFC           0446         MOVLW   B'11111100'    ;CHK IF DIGIT 1
00D2 0186           0447         XORWF   PORT_B,0       ;       /
00D3 0643           0448         BTFSC   STATUS,Z       ;NO THEN SKIP
00D4 0AE5           0449         GOTO    MASK_ALARM     ;ELSE MASK ALARM
00D5 0CF3           0450         MOVLW   B'11110011'    ;CHK IF DIGIT 2
00D6 0186           0451         XORWF   PORT_B,0       ;       /
00D7 0643           0452         BTFSC   STATUS,Z       ;NO THEN SKIP
00D8 0AE8           0453         GOTO    MASK_COLON     ;ELSE MASK COLON
00D9 0CCF           0454         MOVLW   B'11001111'    ;CHK IF DIGIT 3
00DA 0186           0455         XORWF   PORT_B,0       ;       /
00DB 0643           0456         BTFSC   STATUS,Z       ;NO THEN SKIP
00DC 0AE1           0457         GOTO    MASK_PM        ;ELSE MASK PM
                    0458 MASK_AM
00DD 02A4           0459         INCF    FSR            ;INC FSR
00DE 07E0           0460         BTFSS   F0,AM_PM       ;IF 0 THEN AM
00DF 05F2           0461         BSF     DIGIT,7        ;SET MSB
00E0 0AEB           0462         GOTO    BLNK_LEAD_0    ;NEXT
                    0463 MASK_PM
00E1 02A4           0464         INCF    FSR            ;INC FSR
00E2 06E0           0465         BTFSC   F0,AM_PM       ;IF 1 THEN PM
00E3 05F2           0466         BSF     DIGIT,7        ;SET MSB
00E4 0AEB           0467         GOTO    BLNK_LEAD_0    ;NEXT
                    0468 MASK_ALARM
00E5 0650           0469         BTFSC   FLAG,ALRMLED   ;1 THEN LIGHT LED
00E6 05F2           0470         BSF     DIGIT,7        ;       /
00E7 0AEB           0471         GOTO    BLNK_LEAD_0    ;NEXT
                    0472 MASK_COLON
00E8 0670           0473         BTFSC   FLAG,COLON     ;1 THEN LIGHT LED
00E9 05F2           0474         BSF     DIGIT,7        ;       /
00EA 0AEB           0475         GOTO    BLNK_LEAD_0    ;NEXT
                    0476 ;
                    0477 BLNK_LEAD_0
00EB 0210           0478         MOVF    FLAG,W         ;GET IN W
00EC 0E03           0479         ANDLW   B'00000011'    ;SEE IF IN DEM
```

© 1994 Microchip Technology Incorporated

```
00ED 0F02          0480          XORLW   B'00000010'     ;CHECK
00EE 0643          0481          BTFSC   STATUS,Z        ;NO THEN DO
00EF 0800          0482          RETLW   0               ;ELSE RETURN
00F0 0CFC          0483          MOVLW   B'11111100'     ;SEE IF DIGIT 1
00F1 0186          0484          XORWF   PORT_B,0        ;       /
00F2 0743          0485          BTFSS   STATUS,Z        ;YES THEN SKIP
00F3 0800          0486          RETLW   0               ;RETURN
00F4 0C3F          0487          MOVLW   B'00111111'     ;ELSE MASK G AND ANUNC
00F5 0152          0488          ANDWF   DIGIT,0         ;GET IN W
00F6 0F3F          0489          XORLW   B'00111111'     ;SEE IF 0
00F7 0743          0490          BTFSS   STATUS,Z        ;YES THEN SKIP
00F8 0800          0491          RETLW   0               ;RETURN
00F9 0C80          0492          MOVLW   B'10000000'     ;ELSE BLANK D1
00FA 0172          0493          ANDWF   DIGIT           ;       /
00FB 0800          0494          RETLW   0               ;RETURN
                   0495 ;
                   0496 ;
                   0497 ;
                   0499 ;
                   0500 ;THIS ROUTINE SETS UP PORTS A,B,C AND THE INTERNAL
                   0501 ;REAL TIME CLOCK COUNTER.
                   0502 INIT_CLK
00FC 0C0F          0503          MOVLW   B'00001111'     ;MAKE ACTIVE HIGH
00FD 0025          0504          MOVWF   PORT_A          ;       /
00FE 0C00          0505          MOVLW   B'00000000'     ;SET PORT A AS OUTPUTS
00FF 0005          0506          TRIS    PORT_A
                   0507 ;
0100 0CFF          0508          MOVLW   B'11111111'     ;SET LEVELS HIGH
0101 0026          0509          MOVWF   PORT_B          ;       /
0102 0C00          0510          MOVLW   B'00000000'     ;SET PORT B AS OUTPUTS
0103 0006          0511          TRIS    PORT_B
                   0512 ;
0104 0C00          0513          MOVLW   B'00000000'     ;SET LEVELS LOW
0105 0027          0514          MOVWF   PORT_C          ;       /
0106 0C00          0515          MOVLW   B'00000000'     ;SET PORT C AS OUTPUTS
0107 0007          0516          TRIS    PORT_C          ;       /
                   0517 ;
0108 0C04          0518          MOVLW   B'00000100'     ;SET UP PRESCALER
0109 0002          0519          OPTION                  ;       /
                   0520 ;
010A 0C60          0521          MOVLW   MSEC5           ;RTCC = 5 mSEC
010B 0021          0522          MOVWF   RTCC            ;       /
010C 0068          0523          CLRF    MSTMR           ;CLEAR MSTMR
010D 0069          0524          CLRF    STMR            ; & SEC TMR
010E 006A          0525          CLRF    MTMR            ;& MINUTES
010F 0C12          0526          MOVLW   12H             ;MAKE HRS = 12
0110 002B          0527          MOVWF   HTMR            ;       /
0111 002D          0528          MOVWF   HALARM          ;MAKE HRS = 12
0112 006C          0529          CLRF    MALARM          ;       /
0113 0C03          0530          MOVLW   B'00000011'     ;SET TO TEST MODE
0114 0030          0531          MOVWF   FLAG            ;       /
0115 0078          0532          CLRF    ALFLAG          ;CLEAR ALL FLAG
0116 0079          0533          CLRF    AAFLAG          ;       /
0117 0077          0534          CLRF    ENTFLG          ;       /
0118 0A01          0535          GOTO    TEST_HARDWARE
                   0536 ;
                   0537 ;All routines related to timer updates are located at
                   0538 ;address 200 and above.
                   0540          ORG     0200
                   0541 ;
                   0542 UPDATE_TIMERS
0200 0201          0543          MOVF    RTCC,W          ;SEE IF RTCC = 0
0201 0743          0544          BTFSS   STATUS,Z        ;IF 0 THEN SKIP
0202 0A00          0545          GOTO    UPDATE_TIMERS   ;ELSE LOOP
0203 0C60          0546          MOVLW   MSEC5           ;RTCC = 5 mSEC
0204 0021          0547          MOVWF   RTCC            ;       /
0205 02A8          0548          INCF    MSTMR           ;INC 5 MILLI SEC
0206 06D0          0549          BTFSC   FLAG,KEY_HIT    ;NO KEY HIT THEN SKIP
0207 0A70          0550          GOTO    CHK_DE_BOUNCE   ;ELSE DEBOUNCE
```


© 1994 Microchip Technology Incorporated

```
                    0551 UP_TMR_1
0208 0210           0552         MOVF    FLAG,W          ;ALARM MODE?
0209 0E03           0553         ANDLW   B'00000011'     ;     /
020A 0F01           0554         XORLW   B'00000001'     ;     /
020B 0743           0555         BTFSS   STATUS,Z        ;SKIP IF YES
020C 0A14           0556         GOTO    UP_TMR_2        ;DO NEXT
020D 0550           0557         BSF     FLAG,ALRMLED    ;LIGHT LED
020E 0570           0558         BSF     FLAG,COLON      ;     /
020F 0C64           0559         MOVLW   D'100'          ;IF 1/2 SEC
0210 0088           0560         SUBWF   MSTMR,0         ; BLINK
0211 0703           0561         BTFSS   STATUS,C        ;  /
0212 0450           0562         BCF     FLAG,ALRMLED    ;ALARM LED
0213 0A19           0563         GOTO    UP_TMR_3        ;SKIP
                    0564 UP_TMR_2
0214 0570           0565         BSF     FLAG,COLON      ;TURN ON
0215 0C64           0566         MOVLW   D'100'          ;<100 BLINK COLON
0216 0088           0567         SUBWF   MSTMR,0         ;     /
0217 0703           0568         BTFSS   STATUS,C        ;YES THEN SKIP
0218 0470           0569         BCF     FLAG,COLON      ;ELSE TURN OFF
                    0570 UP_TMR_3
0219 0208           0571         MOVF    MSTMR,0         ;GET MSTMR IN W
021A 0FC8           0572         XORLW   D'200'          ;= 200 THEN SKIP
021B 0743           0573         BTFSS   STATUS,Z        ;     /
021C 0800           0574         RETLW   0
                    0575 ;INC SECONDS COUNT
021D 0068           0576         CLRF    MSTMR           ;CLEAR MS_TMR
021E 0216           0577         MOVF    MIN_SEC,W       ;GET MIN_SEC TIMER
021F 0E0F           0578         ANDLW   B'00001111'     ;MASK MINUTES
0220 0743           0579         BTFSS   STATUS,Z        ;ZERO THEN SKIP
0221 00F6           0580         DECF    MIN_SEC         ;REDUCE SECONDS
0222 0C09           0581         MOVLW   STMR            ;LOAD FSR WITH S_TMR
0223 0024           0582         MOVWF   FSR             ;     /
0224 0955           0583         CALL    INC_60          ;INC SECONDS
0225 0D00           0584         IORLW   0               ;DO AN OPERATION
0226 0743           0585         BTFSS   STATUS,Z        ;IF RETURN = 0 SKIP
0227 0A38           0586         GOTO    CHK_AL_TIM      ;CHK ALRM
                    0587 ;INC MINUTES COUNT
0228 03B6           0588         SWAPF   MIN_SEC         ;SWAP MIN SEC
0229 0216           0589         MOVF    MIN_SEC,W       ;GET MIN_SEC IN W
022A 0E0F           0590         ANDLW   B'00001111'     ;MASK SECONDS
022B 0743           0591         BTFSS   STATUS,Z        ;SKIP IF NOT SET
022C 00F6           0592         DECF    MIN_SEC         ;ELSE DEC
022D 03B6           0593         SWAPF   MIN_SEC         ;SWAP BACK
022E 0966           0594         CALL    CHK_SILNC_TIM   ;SILNCE ON?
022F 0C0A           0595         MOVLW   MTMR            ;INC MINUTES
0230 0024           0596         MOVWF   FSR             ;     /
0231 0955           0597         CALL    INC_60          ;     /
0232 0D00           0598         IORLW   0               ;DO AN OPERATION
0233 0743           0599         BTFSS   STATUS,Z        ;IF 0 THEN SKIP
0234 0A38           0600         GOTO    CHK_AL_TIM      ;CHECK ALRAM TIME
                    0601 ;INC HOUR COUNT
0235 0C0B           0602         MOVLW   HTMR            ;GET HTMR IN FSR
0236 0024           0603         MOVWF   FSR
0237 0989           0604         CALL    INC_HR          ;INC HOURS
                    0605 ;
                    0606 CHK_AL_TIM
0238 0718           0607         BTFSS   ALFLAG,ALONOF   ;IF OFF QUIT
0239 0800           0608         RETLW   0               ;     /
023A 0658           0609         BTFSC   ALFLAG,SILNC    ;RET IF IN SILENCE
023B 0800           0610         RETLW   0
023C 0638           0611         BTFSC   ALFLAG,INAL     ;ALREADY DONE
023D 0A4D           0612         GOTO    CHK_1_MIN       ;SEE IF 1 MIN UP
                    0613 ;         RETLW   0             ;YES THEN QUIT
023E 020D           0614         MOVF    HALARM,W        ;CHK HRS
023F 018B           0615         XORWF   HTMR,W          ;EQUAL?
0240 0743           0616         BTFSS   STATUS,Z        ;YES THEN SKIP
0241 0800           0617         RETLW   0               ;ELSE RET
0242 020C           0618         MOVF    MALARM,W        ;CHK MIN
0243 018A           0619         XORWF   MTMR,W          ;EQUAL?
```

```
0244 0743          0620          BTFSS   STATUS,Z        ;YES THEN SKIP
0245 0800          0621          RETLW   0               ;ELSE RET
0246 0209          0622          MOVF    STMR,W          ;SEE IF SEC=0
0247 0743          0623          BTFSS   STATUS,Z        ;YES THEN SKIP
0248 0800          0624          RETLW   0               ;NO THEN RET
0249 0538          0625          BSF     ALFLAG,INAL     ;SET IN ALARM FLAG
024A 0C10          0626          MOVLW   10              ;SET 1 MIN TIMER
024B 0036          0627          MOVWF   MIN_SEC         ;       /
024C 0800          0628          RETLW   0
                   0629 ;
                   0630 CHK_1_MIN
024D 0396          0631          SWAPF   MIN_SEC,W       ;SWAP IN W
024E 0E0F          0632          ANDLW   B'00001111'     ;CHK MINUTES
024F 0743          0633          BTFSS   STATUS,Z        ;0 THEN SKIP
0250 0800          0634          RETLW   0               ;ELSE RET
0251 0438          0635          BCF     ALFLAG,INAL     ;CLR IN ALARM
0252 0478          0636          BCF     ALFLAG,INAA     ;CLR IN AA
0253 0505          0637          BSF     PORT_A,BEP      ;STOP BEEPER
0254 0800          0638          RETLW   0
                   0639 ;
                   0640 INC_60
0255 02A0          0641          INCF    F0              ;INC AND GET IN W
0256 0200          0642          MOVF    F0,0            ;       /
0257 0E0F          0643          ANDLW   B'00001111'     ;MASK HI BITS
0258 0F0A          0644          XORLW   B'00001010'     ;= 10 THEN MAKE IT 0
0259 0743          0645          BTFSS   STATUS,Z        ;       /
025A 0801          0646          RETLW   1               ;ELSE RETURN NON ZERO
025B 0CF0          0647          MOVLW   B'11110000'     ;ZERO LSB
025C 0160          0648          ANDWF   F0              ;       /
025D 03A0          0649          SWAPF   F0              ;SWAP INDIRECT
025E 02A0          0650          INCF    F0              ;INC
025F 0200          0651          MOVF    F0,0            ;GET IN W
0260 03A0          0652          SWAPF   F0              ;SWAP F0 BACK
0261 0F06          0653          XORLW   D'6'            ;=6 THEN SKIP
0262 0743          0654          BTFSS   STATUS,Z        ;       /
0263 0801          0655          RETLW   1               ;ELSE RETURN NZ
0264 0060          0656          CLRF    F0              ;       /
0265 0800          0657          RETLW   0               ;RET 0
                   0658 ;
                   0660 ;
                   0661 CHK_SILNC_TIM
0266 0758          0662          BTFSS   ALFLAG,SILNC    ;CHK IF IN SILENCE
0267 0800          0663          RETLW   0               ;NO THEN SKIP
0268 0396          0664          SWAPF   MIN_SEC,W       ;GET MIN IN W
0269 0E0F          0665          ANDLW   B'00001111'     ;MASK SECS
026A 0743          0666          BTFSS   STATUS,Z        ;ZERO?
026B 0800          0667          RETLW   0               ;NO THEN RET
026C 0458          0668          BCF     ALFLAG,SILNC    ;RESET SILENCE
026D 0C10          0669          MOVLW   10              ;SET I MIN TIMER
026E 0036          0670          MOVWF   MIN_SEC         ;       /
026F 0800          0671          RETLW   0
                   0672 ;
                   0673 ;
                   0674 CHK_DE_BOUNCE
0270 06B7          0675          BTFSC   ENTFLG,INKEYBEP ;IN KEY BEEP?
0271 0A76          0676          GOTO    CHK_DEB_1       ;YES THEN DEC TIMER
0272 07B0          0677          BTFSS   FLAG,KEY_BEEP   ;KEY BEEP SET?
0273 0A7F          0678          GOTO    CHK_SERV        ;NO, SEE IF SERVICED
0274 0678          0679          BTFSC   ALFLAG,INAA     ;IN AA?
0275 0A86          0680          GOTO    CHK_BEP_ON      ;YES THEN SEE IF ON
                   0681 CHK_DEB_1
0276 05B7          0682          BSF     ENTFLG,INKEYBEP ;SET FLAG
0277 0215          0683          MOVF    DEBOUNCE,W      ;GET IN W
0278 0643          0684          BTFSC   STATUS,Z        ;NZ THEN SKIP
0279 0C14          0685          MOVLW   D'20'           ;ELSE DB 100 mSEC
027A 0035          0686          MOVWF   DEBOUNCE        ;       /
027B 0405          0687          BCF     PORT_A,BEP      ;TURN ON BEEPER
027C 02F5          0688          DECFSZ  DEBOUNCE        ;DEC AND CHK
027D 0A08          0689          GOTO    UP_TMR_1        ;GO BACK
```


© 1994 Microchip Technology Incorporated

```
027E 0505          0690          BSF     PORT_A,BEP      ;TURN OFF BEEPER
                   0691 CHK_SERV
                   0692 ;         CLRF    DEBOUNCE
                   0693 ;         BSF     PORT_A,BEP
027F 07F0          0694          BTFSS   FLAG,SERVICED   ;SERVICED THEN SKIP
0280 0A08          0695          GOTO    UP_TMR_1        ;GO BACK
0281 04F0          0696          BCF     FLAG,SERVICED   ;ELSE CLEAR FLAGS
0282 04D0          0697          BCF     FLAG,KEY_HIT    ;
0283 04B0          0698          BCF     FLAG,KEY_BEEP   ;RESET FLAG
0284 04B7          0699          BCF     ENTFLG,INKEYBEP ;
0285 0A08          0700          GOTO    UP_TMR_1        ;GO BACK
                   0701 ;
                   0702 CHK_BEP_ON
0286 0705          0703          BTFSS   PORT_A,BEP      ;IF OFF THEN SKIP
0287 0A08          0704          GOTO    UP_TMR_1        ;ELSE WAIT
0288 0A76          0705          GOTO    CHK_DEB_1       ;RETURN
                   0706 ;
                   0707 ;
                   0708 INC_HR
0289 02A0          0709          INCF    F0              ;INC HOUR TIMER
028A 0200          0710          MOVF    F0,W            ;GET HR TMR IN W
028B 0031          0711          MOVWF   TEMP            ;SAVE IN TEMP
028C 0E0F          0712          ANDLW   B'00001111'     ;CHK LO BYTE = 10
028D 0F0A          0713          XORLW   D'10'           ;
028E 0743          0714          BTFSS   STATUS,Z        ;YES THEN SKIP
028F 0A93          0715          GOTO    INC_AM_PM       ;ELSE CHK 12
0290 0C10          0716          MOVLW   B'00010000'     ;LOAD 1 IN MSB
0291 0020          0717          MOVWF   F0
0292 0AA3          0718          GOTO    RESTORE_AM_PM   ;RESTORE AM/PM
                   0719 INC_AM_PM
0293 04E0          0720          BCF     F0,AM_PM        ;CLEAR AM/PM
0294 0200          0721          MOVF    F0,W            ;GET IN W
0295 0F12          0722          XORLW   12H             ;SEE IF 12 HEX
0296 0743          0723          BTFSS   STATUS,Z        ;YES THEN SKIP
0297 0A9D          0724          GOTO    CHK_13          ;ELSE CHK 13
0298 07F1          0725          BTFSS   TEMP,AM_PM      ;IF SET, SKIP
0299 0A9C          0726          GOTO    SET_AM_PM       ;ELSE SET
029A 04E0          0727          BCF     F0,AM_PM        ;CLEAR FLAG
029B 0800          0728          RETLW   0               ;RETURN
                   0729 SET_AM_PM
029C 05E0          0730          BSF     F0,AM_PM        ;SET FLAG
                   0731 CHK_13
029D 0200          0732          MOVF    F0,W            ;GET IN W
029E 0F13          0733          XORLW   13H             ;SEE IF 13
029F 0743          0734          BTFSS   STATUS,Z        ;YES THEN SKIP
02A0 0AA3          0735          GOTO    RESTORE_AM_PM
                   0736 SET_1_HR
02A1 0C01          0737          MOVLW   B'00000001'     ;SET TO 1
02A2 0020          0738          MOVWF   F0
                   0739 RESTORE_AM_PM
02A3 06F1          0740          BTFSC   TEMP,AM_PM      ;SKIP IF AM
02A4 05E0          0741          BSF     F0,AM_PM        ;ELSE SET TO PM
02A5 0800          0742          RETLW   0
                   0743 ;
                   0744 ;
                   0745 ;
                   0747          ORG     400
                   0748 ;
                   0749 ;KEY DEFINITIONS
000A               0750          ALARM_KEY       EQU     0A
000B               0751          CE_KEY          EQU     0B
000C               0752          SNOOZE_KEY      EQU     0C
000D               0753          AM_PM_KEY       EQU     0D
000E               0754          CLR_ALARM_KEY   EQU     0E
000F               0755          SET_KEY         EQU     0F
                   0756 ;
                   0757 SERVICE_KEYS
0400 07D0          0758          BTFSS   FLAG,KEY_HIT    ;NO KEY HIT THEN ...
0401 0800          0759          RETLW   0               ;RETURN
```

© 1994 Microchip Technology Incorporated

```
0402 06F0          0760          BTFSC   FLAG,SERVICED   ;IF NOT SERVICED SKIP
0403 0800          0761          RETLW   0               ;ELSE RETURN
0404 05F0          0762          BSF     FLAG,SERVICED   ;SET SERVICED FLAG
0405 0210          0763          MOVF    FLAG,W          ;GET MODE OF OPERATION
0406 0E03          0764          ANDLW   B'00000011'     ;      /
0407 0643          0765          BTFSC   STATUS,Z        ;00 THEN RTM
0408 0A10          0766          GOTO    RTMKS           ;RTM KEY SERVICE
0409 0031          0767          MOVWF   TEMP            ;SAVE IN TEMP
040A 02F1          0768          DECFSZ  TEMP            ;REDUCE TEMP
040B 0A0D          0769          GOTO    SK1             ;SKIP
040C 0A1D          0770          GOTO    ATMKS           ;01, DO ALARM MODE
                   0771 SK1
040D 02F1          0772          DECFSZ  TEMP            ;REDUCE TEMP
040E 0800          0773          RETLW   0               ;11 THEN RETURN
040F 0A2A          0774          GOTO    DEMKS           ;10, DATA ENTRY MODE
                   0775 ;
                   0776 ;REAL TIME MODE KEY SERVICE
                   0777 RTMKS
0410 09BA          0778          CALL    CHK_AL_KEYS     ;CHK ALARM KEYS
0411 0D00          0779          IORLW   0               ;SEE IF NZ RET
0412 0643          0780          BTFSC   STATUS,Z        ;NZ THEN SKIP
0413 0800          0781          RETLW   0               ;ELSE RETURN
0414 0C0F          0782          MOVLW   SET_KEY         ;SEE IF SET KEY
0415 0193          0783          XORWF   NEW_KEY,W       ;      /
0416 0643          0784          BTFSC   STATUS,Z        ;NO THEN SKIP
0417 0A91          0785          GOTO    SERV_SET_RTM    ;SERVICE SET KEY
0418 0C0A          0786          MOVLW   ALARM_KEY       ;ALARM KEY?
0419 0193          0787          XORWF   NEW_KEY,W       ;      /
041A 0643          0788          BTFSC   STATUS,Z        ;NO THEN SKIP
041B 0AAB          0789          GOTO    SERV_ALARM_RTM  ;ELSE SERVICE ALARM
                   0790 IGNORE_KEY
041C 0800          0791          RETLW   0               ;ELSE RETURN
                   0792 ;
                   0793 ;ALARM TIME MODE KEY SERVICE
                   0794 ATMKS
041D 09BA          0795          CALL    CHK_AL_KEYS     ;CHECK ALRM KEYS
041E 0D00          0796          IORLW   0               ;CHECK IF 0
041F 0643          0797          BTFSC   STATUS,Z        ;NZ THEN SKIP
0420 0800          0798          RETLW   0               ;ELSE RETURN
0421 0C0F          0799          MOVLW   SET_KEY         ;SEE IF SET KEY
0422 0193          0800          XORWF   NEW_KEY,W       ;      /
0423 0643          0801          BTFSC   STATUS,Z        ;NO THEN SKIP
0424 0A9C          0802          GOTO    SERV_SET_ATM
0425 0C0A          0803          MOVLW   ALARM_KEY       ;GET ALARM KEY
0426 0193          0804          XORWF   NEW_KEY,W       ;SEE IF HIT
0427 0643          0805          BTFSC   STATUS,Z        ;NO THEN SKIP
0428 0AA2          0806          GOTO    SERV_ALARM_ATM  ;ELSE SERVICE
0429 0A1C          0807          GOTO    IGNORE_KEY
                   0808 ;
                   0809 ;DATA ENTRY MODE KEY SERVICE
                   0810 DEMKS
042A 09BA          0811          CALL    CHK_AL_KEYS     ;CHECK ALARM KEYS
042B 0D00          0812          IORLW   0               ;CHK IF 0
042C 0643          0813          BTFSC   STATUS,Z        ;NZ THEN SKIP
042D 0800          0814          RETLW   0               ;ELSE RETURN
042E 0C0F          0815          MOVLW   SET_KEY         ;IF SET KEY THEN END
042F 0193          0816          XORWF   NEW_KEY,W       ;      /
0430 0643          0817          BTFSC   STATUS,Z        ;NO THEN SKIP
0431 0A3F          0818          GOTO    DEMKS_END       ;GOTO END
0432 0C0B          0819          MOVLW   CE_KEY          ;IF CLEAR ENTRY
0433 0193          0820          XORWF   NEW_KEY,W       ;      /
0434 0643          0821          BTFSC   STATUS,Z        ;SKIP IF NO
0435 0A48          0822          GOTO    DEMKS_END_1     ;ABANDON ENTRY
0436 0737          0823          BTFSS   ENTFLG,HR10     ;10'S HRS DONE?
0437 0A54          0824          GOTO    ENT_HR_10       ;NO THEN GET
0438 0757          0825          BTFSS   ENTFLG,HR       ;HRS DONE?
0439 0A5F          0826          GOTO    ENT_HRS         ;NO THEN GET
043A 0777          0827          BTFSS   ENTFLG,MIN10    ;10'S MIN. DONE?
043B 0A72          0828          GOTO    ENT_MIN_10      ;NO THEN GET
```


© 1994 Microchip Technology Incorporated

```
043C 0797          0829          BTFSS   ENTFLG,MIN      ;MIN DONE?
043D 0A7F          0830          GOTO    ENT_MIN         ;NO THEN GET
043E 0A87          0831          GOTO    ENT_AM_PM       ;NO THEN GET
                   0832 DEMKS_END
043F 0717          0833          BTFSS   ENTFLG,RTATS    ;GET OLD STATUS
0440 0A4D          0834          GOTO    LD_RTM          ;LOAD IN TIME
0441 020E          0835          MOVF    MENTRY,W        ;LD IN ALARM
0442 002C          0836          MOVWF   MALARM          ;      /
0443 020F          0837          MOVF    HENTRY,W        ;     /
0444 002D          0838          MOVWF   HALARM          ;    /
0445 0450          0839          BCF     FLAG,ALRMLED    ;CLEAR FLAG
0446 0618          0840          BTFSC   ALFLAG,ALONOF   ;SEE IF ON-OFF
0447 0550          0841          BSF     FLAG,ALRMLED    ;ELSE SET
                   0842 DEMKS_END_1
0448 0410          0843          BCF     FLAG,0          ;RTM MODE
0449 0430          0844          BCF     FLAG,1          ;     /
044A 0490          0845          BCF     FLAG,FLASH      ;STOP FLASH
                   0846 SERV_COM_RET
044B 05B0          0847          BSF     FLAG,KEY_BEEP
044C 0800          0848          RETLW   0               ;RETURN
                   0849 ;
                   0850 LD_RTM
044D 020E          0851          MOVF    MENTRY,W        ;LD IN RTM
044E 002A          0852          MOVWF   MTMR            ;      /
044F 020F          0853          MOVF    HENTRY,W        ;     /
0450 002B          0854          MOVWF   HTMR            ;    /
0451 0068          0855          CLRF    MSTMR           ;CLR TIME
0452 0069          0856          CLRF    STMR            ;  /
0453 0A48          0857          GOTO    DEMKS_END_1     ;GO BACK
                   0858 ;
                   0859 ENT_HR_10
0454 0213          0860          MOVF    NEW_KEY,W       ;SEE IF 0
0455 0643          0861          BTFSC   STATUS,Z        ;NZ THEN SKIP
0456 0A5C          0862          GOTO    LD_HENTRY_0     ;LOAD 0
0457 02D3          0863          DECFSZ  NEW_KEY,0       ;1 THE SKIP
0458 0A1C          0864          GOTO    IGNORE_KEY      ;ELSE IGNORE KEY
0459 058F          0865          BSF     HENTRY,4        ;SET TO 1
045A 0537          0866          BSF     ENTFLG,HR10     ;SET FLAG
045B 0A4B          0867          GOTO    SERV_COM_RET    ;GO GET NEXT
                   0868 LD_HENTRY_0
045C 048F          0869          BCF     HENTRY,4        ;SET TO 0
045D 0537          0870          BSF     ENTFLG,HR10
045E 0A4B          0871          GOTO    SERV_COM_RET    ;
                   0872 ENT_HRS
045F 0C0F          0873          MOVLW   HENTRY          ;USE INDIRECT ADDR.
0460 0024          0874          MOVWF   FSR             ;      /
0461 068F          0875          BTFSC   HENTRY,4        ;SEE IF 0
0462 0A6D          0876          GOTO    ALLOW0_2        ;YES THEN 0,1&2
0463 0C0A          0877          MOVLW   D'10'           ;SEE IF 0 - 9
0464 0093          0878          SUBWF   NEW_KEY,W       ;      /
0465 0603          0879          BTFSC   STATUS,C        ;IF C THEN SKIP
0466 0A1C          0880          GOTO    IGNORE_KEY      ;ELSE IGNORE
                   0881 ENT_LO_COM1
0467 0557          0882          BSF     ENTFLG,HR       ;SET FLAG
                   0883 ENT_LO_COM
0468 0200          0884          MOVF    F0,W            ;LD HRS
0469 0EF0          0885          ANDLW   B'11110000'     ;MASK LO NIBL
046A 0113          0886          IORWF   NEW_KEY,W       ;OR NEW KEY
046B 0020          0887          MOVWF   F0              ;SAVE BACK
046C 0A4B          0888          GOTO    SERV_COM_RET    ;GET NEXT
                   0889 ALLOW0_2
046D 0C03          0890          MOVLW   D'3'            ;SEE IF 0 - 2
046E 0093          0891          SUBWF   NEW_KEY,W       ;      /
046F 0603          0892          BTFSC   STATUS,C        ;<3 THEN SKIP
0470 0A1C          0893          GOTO    IGNORE_KEY
0471 0A67          0894          GOTO    ENT_LO_COM1     ;     /
                   0895 ;
                   0896 ENT_MIN_10
0472 0C0E          0897          MOVLW   MENTRY          ;DO INDIRECT ADDR.
```

© 1994 Microchip Technology Incorporated

```
0475 0093          0900          SUBWF   NEW_KEY,W       ;     /
0476 0603          0901          BTFSC   STATUS,C        ;IF C THEN SKIP
0477 0A1C          0902          GOTO    IGNORE_KEY      ;ELSE IGNORE
0478 0380          0903          SWAPF   F0,W            ;SWAP AND GET
0479 0EF0          0904          ANDLW   B'11110000'     ;MASK LO NIBL
047A 0113          0905          IORWF   NEW_KEY,W       ;OR NEW KEY
047B 0020          0906          MOVWF   F0              ;SAVE BACK
047C 03A0          0907          SWAPF   F0              ;SWAP BACK
047D 0577          0908          BSF     ENTFLG,MIN10
047E 0A4B          0909          GOTO    SERV_COM_RET    ;GET NEXT
                   0910 ;
                   0911 ENT_MIN
047F 0C0E          0912          MOVLW   MENTRY          ;DO INDIRECT
0480 0024          0913          MOVWF   FSR             ;     /
0481 0C0A          0914          MOVLW   D'10'           ;ALLOW 0 - 9
0482 0093          0915          SUBWF   NEW_KEY,W       ;SEE IF >
0483 0603          0916          BTFSC   STATUS,C        ;NO THEN SKIP
0484 0A1C          0917          GOTO    IGNORE_KEY      ;ELSE IGNORE
0485 0597          0918          BSF     ENTFLG,MIN      ;SET FLAG
0486 0A68          0919          GOTO    ENT_LO_COM      ;     /
                   0920 ;
                   0921 ENT_AM_PM
0487 0C0D          0922          MOVLW   AM_PM_KEY       ;AM/PM KEY?
0488 0193          0923          XORWF   NEW_KEY,W       ;     /
0489 0743          0924          BTFSS   STATUS,Z        ;YES THEN SKIP
048A 0A1C          0925          GOTO    IGNORE_KEY
048B 07EF          0926          BTFSS   HENTRY,AM_PM    ;TEST BIT
048C 0A8F          0927          GOTO    SETAMPM         ;ELSE SET
048D 04EF          0928          BCF     HENTRY,AM_PM    ;CLEAR FLAG
048E 0A4B          0929          GOTO    SERV_COM_RET    ;GOTO END
                   0930 SETAMPM
048F 05EF          0931          BSF     HENTRY,AM_PM    ;SET FLAG
0490 0A4B          0932          GOTO    SERV_COM_RET
                   0933 ;
                   0934 ;
                   0935 ;
                   0936 SERV_SET_RTM
0491 020A          0937          MOVF    MTMR,W          ;TRANSFER TIME
0492 002E          0938          MOVWF   MENTRY          ;TO DATA ENTRY
0493 020B          0939          MOVF    HTMR,W          ;     /
0494 002F          0940          MOVWF   HENTRY          ;    /
                   0941 SERV_COM
0495 0210          0942          MOVF    FLAG,W          ;SAVE IN W
0496 0E01          0943          ANDLW   B'00000001'     ;ATM OR RTM MODE?
0497 0037          0944          MOVWF   ENTFLG          ;SAVE IN ENTFLG
0498 0CF2          0945          MOVLW   B'11110010'     ;FORCE 1S
0499 0130          0946          IORWF   FLAG            ;     /
049A 0410          0947          BCF     FLAG,0          ;     /
049B 0800          0948          RETLW   0
                   0949 ;
                   0950 SERV_SET_ATM
049C 020C          0951          MOVF    MALARM,W        ;TRANSFER ALARM
049D 002E          0952          MOVWF   MENTRY          ;TO DATA ENTRY
049E 020D          0953          MOVF    HALARM,W        ;     /
049F 002F          0954          MOVWF   HENTRY          ;    /
04A0 0518          0955          BSF     ALFLAG,ALONOF   ;SET FLAG
04A1 0A95          0956          GOTO    SERV_COM        ;GOTO COMMON
                   0957 ;
                   0958 SERV_ALARM_ATM
04A2 0718          0959          BTFSS   ALFLAG,ALONOF   ;TEST ON/OFF
04A3 0AA6          0960          GOTO    SET_ALONOF      ;SET ON/OF FLG
04A4 0418          0961          BCF     ALFLAG,ALONOF   ;CLEAR FLAG
04A5 0AA7          0962          GOTO    SERV_ATM_COM    ;RET THRO COM
                   0963 SET_ALONOF
04A6 0518          0964          BSF     ALFLAG,ALONOF   ;SET FLAG
                   0965 SERV_ATM_COM
04A7 05B0          0966          BSF     FLAG,KEY_BEEP   ;BEEP
```


© 1994 Microchip Technology Incorporated

```
04A8 0CF0          0967          MOVLW   B'11110000'     ;CLEAR SEC COUNT
04A9 0176          0968          ANDWF   MIN_SEC         ;       /
04AA 0800          0969          RETLW   0               ;RETURN
                   0970 ;
                   0971 SERV_ALARM_RTM
04AB 05B0          0972          BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04AC 0510          0973          BSF     FLAG,0          ;SET TO ALARM TIME
04AD 0430          0974          BCF     FLAG,1          ;       /
04AE 0C05          0975          MOVLW   D'05'           ;SAVE 5 IN MIN_SEC
04AF 0036          0976          MOVWF   MIN_SEC         ;       /
04B0 0800          0977          RETLW   0
                   0978 ;
                   0979 SERV_SNOOZE
04B1 0CA0          0980          MOVLW   0A0             ;SNOOZE FOR 10 MINS
04B2 0036          0981          MOVWF   MIN_SEC         ;       /
04B3 0558          0982          BSF     ALFLAG,SILNC    ;SET FLAG
                   0983 CLR_AL_COM
04B4 05B0          0984          BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04B5 007A          0985          CLRF    AATMR           ;RESET AA TIMER
04B6 0079          0986          CLRF    AAFLAG          ;CLEAR AA FLAGS
04B7 0478          0987          BCF     ALFLAG,INAA     ;RESET INAA FLAG
04B8 0505          0988          BSF     PORT_A,BEP      ;TURN OFF BEEPER
04B9 0800          0989          RETLW   0               ;RET
                   0990 ;
                   0991 CHK_AL_KEYS
04BA 0718          0992          BTFSS   ALFLAG,ALONOF   ;ALARM ON?
04BB 0801          0993          RETLW   1               ;NO THEN RET
04BC 0738          0994          BTFSS   ALFLAG,INAL     ;IN ALARM?
04BD 0801          0995          RETLW   1               ;NO THEN SKIP
04BE 0C0E          0996          MOVLW   CLR_ALARM_KEY   ;CHECK IF CLR ALARM
04BF 0193          0997          XORWF   NEW_KEY,W       ;       /
04C0 0643          0998          BTFSC   STATUS,Z        ;NO THEN SKIP
04C1 0AC7          0999          GOTO    CLR_ALARM       ;ELSE CLEAR ALARM
04C2 0C0C          1000          MOVLW   SNOOZE_KEY      ;SEE IF SNOOZE HIT
04C3 0193          1001          XORWF   NEW_KEY,W       ;       /
04C4 0743          1002          BTFSS   STATUS,Z        ;YES THEN SKIP
04C5 0801          1003          RETLW   1
04C6 0AB1          1004          GOTO    SERV_SNOOZE
                   1005 ;
                   1006 CLR_ALARM
04C7 0438          1007          BCF     ALFLAG,INAL     ;CLEAR ALARM
04C8 0458          1008          BCF     ALFLAG,SILNC    ;CLEAR SILENCE
04C9 0C0F          1009          MOVLW   B'00001111'     ;CLEAR MINS
04CA 0176          1010          ANDWF   MIN_SEC         ;       /
04CB 0AB4          1011          GOTO    CLR_AL_COM
                   1012 ;
                   1013          ORG     600
                   1014 ;If the AA alarm is set, then this routine takes care of
                   1015 ;the timing in sounding the alarm.
                   1016 ;
                   1017 SOUND_AA
0600 0738          1018          BTFSS   ALFLAG,INAL     ;SKIP IF IN ALRM
0601 0800          1019          RETLW   0               ;ELSE RETURN
0602 0658          1020          BTFSC   ALFLAG,SILNC    ;SKIP IF NOT IN SIL
0603 0800          1021          RETLW   0               ;ELSE RET
0604 06B7          1022          BTFSC   ENTFLG,INKEYBEP ;SKIP IF NOT IN KEY BEP
0605 0A55          1023          GOTO    CHK_COLSN       ;CHK COLLISION
                   1024 SND_AA_0
0606 0778          1025          BTFSS   ALFLAG,INAA     ;SKIP IF IN AA
                   1026 SND_AA_1
0607 0919          1027          CALL    INIT_AA         ;INIT ALL
0608 0719          1028          BTFSS   AAFLAG,0        ;SKIP IF DONE
0609 0A21          1029          GOTO    DO_CYCL0        ;DO FIRST CYCL
060A 0739          1030          BTFSS   AAFLAG,1        ;SKIP IF DONE
060B 0A29          1031          GOTO    DO_CYCL1        ;ELSE 2ND CYCLE
060C 0759          1032          BTFSS   AAFLAG,2        ;SKIP IF DONE
060D 0A31          1033          GOTO    DO_CYCL2        ;ELSE DO 3RD CYCLE
060E 0779          1034          BTFSS   AAFLAG,3        ;SKIP IF DONE
060F 0A39          1035          GOTO    DO_CYCL3        ;DO CYCLE 4
```

© 1994 Microchip Technology Incorporated

```
0610 0799          1036          BTFSS   AAFLAG,4        ;SKIP IF DONE
0611 0A3E          1037          GOTO    DO_CYCL4        ;DO CYCLE 5
0612 07B9          1038          BTFSS   AAFLAG,5        ;SKIP IF DONE
0613 0A43          1039          GOTO    DO_CYCL5        ;DO CYCLE 6
0614 07D9          1040          BTFSS   AAFLAG,6        ;SKIP IF DONE
0615 0A48          1041          GOTO    DO_CYCL6        ;DO CYCLE 6
0616 07F9          1042          BTFSS   AAFLAG,7        ;SKIP IF DONE
0617 0A50          1043          GOTO    DO_CYCL7        ;DO CYCLE 7
0618 0A07          1044          GOTO    SND_AA_1        ;GO BACK
                   1045 ;
                   1046 INIT_AA
0619 0079          1047          CLRF    AAFLAG          ;CLEAR ALL FLAGS
061A 0578          1048          BSF     ALFLAG,INAA     ;SET IN AA FLAG
061B 0A2D          1049          GOTO    PUT_ON_100      ;ON 100 MSECS
                   1050 ;
                   1051 DEC_AA_TMR
061C 00FA          1052          DECF    AATMR           ;REDUCE TIMER
061D 021A          1053          MOVF    AATMR,W         ;GET IN W
061E 0743          1054          BTFSS   STATUS,Z        ;CHECK IF Z
061F 0801          1055          RETLW   1               ;NO THEN NZ
0620 0800          1056          RETLW   0               ;ELSE 0
                   1057 ;
                   1058 DO_CYCL0
0621 091C          1059          CALL    DEC_AA_TMR      ;REDUCE TIMER
0622 0743          1060          BTFSS   STATUS,Z        ;IF NZ THEN RET
0623 0800          1061          RETLW   0
0624 0519          1062          BSF     AAFLAG,0        ;SET DONE FLAG
                   1063 PUT_OFF_100
0625 0505          1064          BSF     PORT_A,BEP      ;TURN OFF BEEPER
0626 0C14          1065          MOVLW   D'20'           ;FOR 100 MSECS
0627 003A          1066          MOVWF   AATMR           ;     /
0628 0800          1067          RETLW   0
                   1068 ;
                   1069 DO_CYCL1
0629 091C          1070          CALL    DEC_AA_TMR      ;REDUCE TIMER
062A 0743          1071          BTFSS   STATUS,Z        ;IF NZ THEN RET
062B 0800          1072          RETLW   0
062C 0539          1073          BSF     AAFLAG,1        ;SET DONE FLAG
                   1074 PUT_ON_100
062D 0405          1075          BCF     PORT_A,BEP      ;TURN ON BEEPER
062E 0C14          1076          MOVLW   D'20'           ;FOR 100 MSECS
062F 003A          1077          MOVWF   AATMR           ;     /
0630 0800          1078          RETLW   0
                   1079 ;
                   1080 DO_CYCL2
0631 091C          1081          CALL    DEC_AA_TMR      ;REDUCE TIMER
0632 0743          1082          BTFSS   STATUS,Z        ;IF NZ THEN RET
0633 0800          1083          RETLW   0               ;     /
0634 0559          1084          BSF     AAFLAG,2        ;SET DONE FLAG
0635 0505          1085          BSF     PORT_A,BEP      ;TURN OFF BEEPER
0636 0C64          1086          MOVLW   D'100'          ;FOR 500 MSECS
0637 003A          1087          MOVWF   AATMR           ;     /
0638 0800          1088          RETLW   0
                   1089 ;
                   1090 DO_CYCL3
0639 091C          1091          CALL    DEC_AA_TMR      ;REDUCE TIMER
063A 0743          1092          BTFSS   STATUS,Z        ;IF NZ THEN RET
063B 0800          1093          RETLW   0               ;     /
063C 0579          1094          BSF     AAFLAG,3        ;SET DONE FLAG
063D 0A2D          1095          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   1096 ;
                   1097 DO_CYCL4
063E 091C          1098          CALL    DEC_AA_TMR      ;REDUCE TIMER
063F 0743          1099          BTFSS   STATUS,Z        ;IF NZ THEN RET
0640 0800          1100          RETLW   0               ;     /
0641 0599          1101          BSF     AAFLAG,4        ;SET DONE FLAG
0642 0A25          1102          GOTO    PUT_OFF_100     ;DO NEXT CYCLE
                   1103 ;
                   1104 DO_CYCL5
```


© 1994 Microchip Technology Incorporated

```
0643 091C          1105          CALL    DEC_AA_TMR      ;REDUCE TIMER
0644 0743          1106          BTFSS   STATUS,Z       ;IF NZ THEN RET
0645 0800          1107          RETLW   0               ;       /
0646 05B9          1108          BSF     AAFLAG,5        ;SET DONE FLAG
0647 0A2D          1109          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   1110 ;
                   1111 DO_CYCL6
0648 091C          1112          CALL    DEC_AA_TMR      ;REDUCE TIMER
0649 0743          1113          BTFSS   STATUS,Z       ;IF NZ THEN RET
064A 0800          1114          RETLW   0               ;       /
064B 05D9          1115          BSF     AAFLAG,6        ;SET DONE FLAG
064C 0505          1116          BSF     PORT_A,BEP      ;TURN OFF BEEPER
064D 0CC8          1117          MOVLW   D'200'          ;FOR 1000 MSECS
064E 003A          1118          MOVWF   AATMR           ;       /
064F 0800          1119          RETLW   0
                   1120 ;
                   1121 DO_CYCL7
0650 091C          1122          CALL    DEC_AA_TMR      ;REDUCE TIMER
0651 0743          1123          BTFSS   STATUS,Z       ;IF NZ THEN RET
0652 0800          1124          RETLW   0               ;       /
0653 05F9          1125          BSF     AAFLAG,7        ;SET DONE FLAG
0654 0A2D          1126          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   1127 ;
                   1128 CHK_COLSN
0655 0605          1129          BTFSC   PORT_A,BEP      ;IF ON THEN SKIP
0656 0A06          1130          GOTO    SND_AA_0        ;ELSE RET
0657 021A          1131          MOVF    AATMR,W         ;GET TIMER
0658 0643          1132          BTFSC   STATUS,Z        ;NZ THEN SKIP
0659 0A5C          1133          GOTO    LD_AAT_1        ;LOAD A 1 IN TMR
065A 00FA          1134          DECF    AATMR           ;REDUCE TIMER
065B 0800          1135          RETLW   0               ;RETURN
                   1136 LD_AAT_1
065C 02BA          1137          INCF    AATMR           ;INC TIMER
065D 0800          1138          RETLW   0               ;RET
                   1139 ;
                   1140          ORG     PIC57
                   1141 SYS_RESET
07FF 0A00          1142          GOTO    START
                   1143 ;
                   1144          END
                   1145
                   1146
                   1147
                   1148
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXX XXXXXXXXX--- -------- --------
0140 : -------- -------- -------- --------

0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX------ --------
02C0 : -------- -------- -------- --------

0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
04C0 : XXXXXXXXXXXX---- -------- -------- --------
```

© 1994 Microchip Technology Incorporated

```
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ________ ________

0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ _______X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Incorporated

Using PIC16C5X Microcontrollers as LCD Drivers

INTRODUCTION

This application report describes an LCD controller implementation using a PIC16C55 microcontroller. This technique offers display capabilities for applications that require a small display at a low cost together with the capabilities of the standard PIC16C55 microcontroller. We start by an overview of LCD devices and their theory of operation followed by software implementation issues of the controller. The source code for controlling a multiplexed LCD display is included in Appendix A.

LIQUID CRYSTAL DISPLAYS

The Liquid Crystal Display (LCD) is a thin layer of "Liquid Crystal Material" deposited between two plates of glass. The raw LCD is often referred to as "glass". Electrodes are attached to both sides of the glass. One side is referred to as common or backplane, while the other side is referred to as segment.

An LCD is modelled as a capacitor, with one side connected to the common plane and the other side connected to the segment, as shown in Figure 1. LCDs are sensitive to Root Mean Square Voltage levels. When a V_{RMS} level of zero volts is applied to the LCD, the LCD is practically transparent.

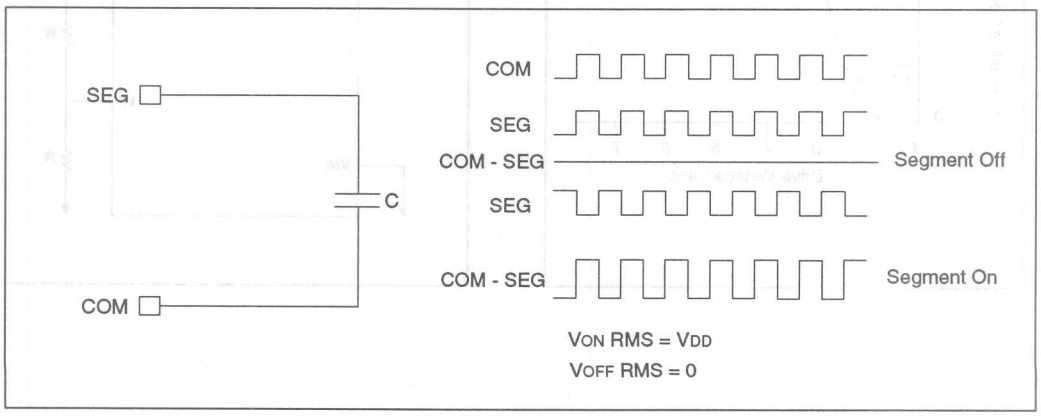
To turn an LCD segment "on", which makes the segment turn dark or opaque, an LCD RMS voltage that is greater than the LCD threshold voltage is applied to the LCD. The RMS LCD voltage is the RMS voltage across the capacitor C in Figure 1, which is equal to the potential difference between SEG and COM values.

Different LCDs have different characteristics; Figure 2 shows typical voltage vs relative contrast characteristics. Notation on curve shows operating points for multiplex operation with the threshold voltage set to 1.7 Vrms. This voltage is often used as the measure of voltage for LCD to be "off" or transparent. The curve is normalized and assumes a viewing angle of 90° to the plane of the LCD.

Contrast control, the process of turning on a segment, is achieved by moving the operating point of the LCD by applying voltage to the LCD that is greater than the LCD threshold voltages. A typical circuit to accomplish this task is shown in Figure 3.

Driving a liquid crystal display at direct current (DC) will cause permanent damage to the display unit. In order to prevent irreversible electrochemical action from destroying the display, the voltage at all segment locations must reverse polarity periodically so that a zero net voltage is applied to the device. This process is referred to as AC voltage application. There are two LCD driving methods available: Static driving method and multiplexed driving method.

FIGURE 1: ELECTRICAL MODEL OF AN LCD SEGMENT WITH DRIVING VOLTAGES



© 1993 Microchip Technology Inc.

Conventional LCDs have separate external connections for each and every segment plus a common plane. This is the most basic method that results in good display quality. The main disadvantage of this driving method is that each segment requires one liquid crystal driver. The static driving method uses the frame frequency, defined as a period of the common plane signal, of several tens to several hundred Hz. A lower frequency would result in blinking effects and higher frequencies would increase power requirements. To turn a segment on, a voltage that has an opposite polarity to the common plane signal must be applied resulting in a large RMS voltage across the plates. To turn off a segment a voltage that is of the same polarity to common plane signal is applied. This drive method is universal to driving LCD segments. Figure 1 shows an example of this driving method.

The LCD frequency is defined as the rate of output changes of the common plane and segment signals, whereas the frame rate is defined as $f_{frame} = \frac{f_{frame}}{N}$ where N is the multiplex rate or number of backplane. Typically, f_{frame} ranges from 25HZ to 300HZ. The most commonly used frame fequency is 40-70HZ. A lower frequency would result in flicker effects and higher frequency would increase power requirements.

Multiplexed LCDs maintain their liquid crystal characteristics. These are low power consumption, high contrast ratio under high ambient light levels, and reduce the number of external connections necessary for dot matrix and alphanumeric displays. The multiplex driving method reduces the number of driver circuits, or microcontroller I/O pins if a software method is used. The method of drive for multiplexed displays is Time Division Multiplex (TDM) with the number of time divisions equal to twice the number of common planes used in a given format. In order to prevent permanent damage to the LCD display, the voltage at all segment locations must reverse polarity periodically so that zero net voltage is applied. This is the reason for the doubling in time divisions; each common plane must be alternately driven with a voltage pulse of opposite polarity. The drive frequency should be greater than the flicker rate of 25 Hz. Since increasing the drive frequency significantly above this value increases current demand by the CMOS circuitry, an upper drive frequency level of 60 Hz is recommended by most LCD manufacturers. We have chosen a drive rate of 50 Hz for this application report which results in a frame period of 20 ms. The most commonly available formats are 2x4, 3x3, and 5x7. In this report we use a 2x4 format LCD to display hexadecimal digits.

FIGURE 2: TYPICAL LCD CHARACTERISTICS

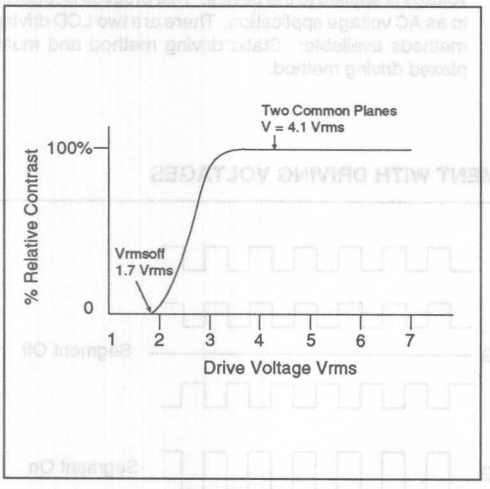
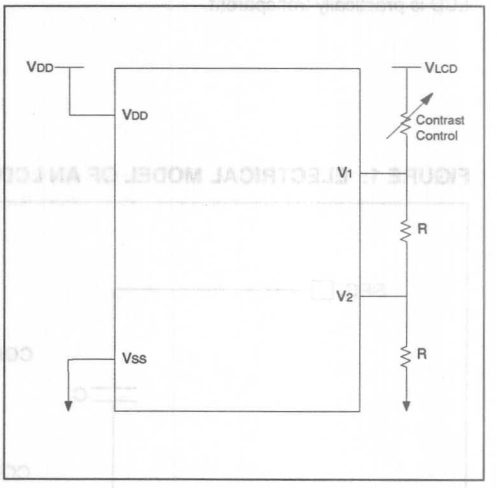


FIGURE 3: CONTRAST CONTROL CIRCUIT



© 1993 Microchip Technology Inc.

To better understand multiplexed LCD control it is best to look at the general case. The segments in a multiplexed LCD are arranged in an X-Y grid form as shown in Figure 4. The common plane signals maintain their relative shape at all times, as shown in Figure 5. To turn on segment 1 (SEG1), we need to apply a voltage Vd, such that Vs+Vd turns the segment on and Vs-Vd turns the segment off. Note that the segment signal Vd is symmetrical. This is a consequence of the intervals that the common plane signal is not present at all times. Use of nonsymmetrical waveform will result in a higher Vrms present on the unaddressed segments. The symmetrical nature of the waveforms theoretically result in a zero DC voltage levels. CMOS drivers (e.g. microcontrollers) operate at 0 to +5V levels (rail voltage levels). This would require driving voltages beyond the range of operation. This constraint is addressed by a technique referred to as "level shifting" or "biasing". Level shifting allows application of voltages in the range of 0 to +2.5V, which is compatible with these drivers. This would require an additional voltage level of +2.5V, which can be implemented through a simple resistive voltage divider circuit.

2

FIGURE 4: MULTIPLEXED LCD SEGMENT ARRANGEMENT

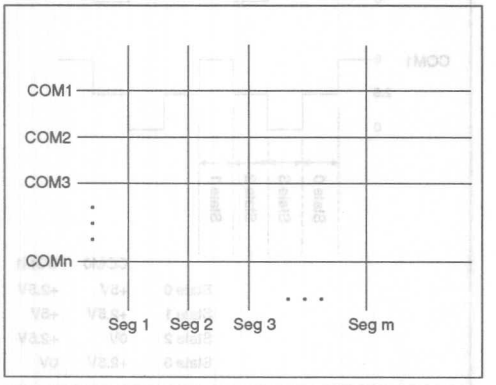
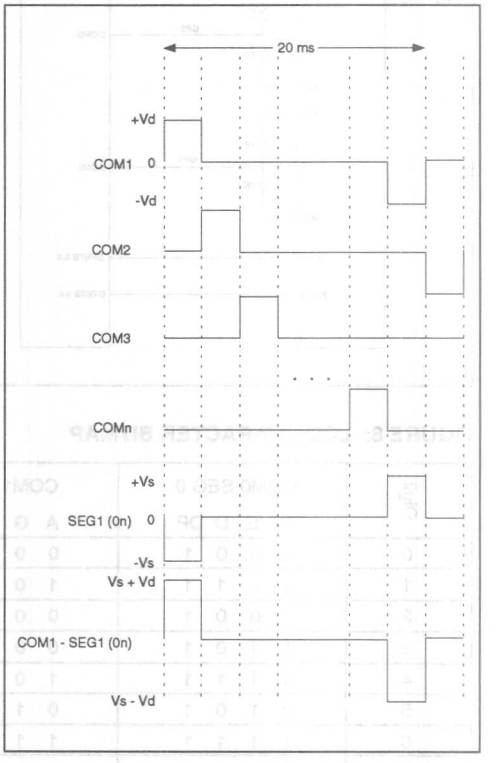


FIGURE 5: MULTIPLEXED LCD DRIVE WAVEFORMS



The ideas presented in the previous section can be applied to any size multiplexed LCD display. In our implementation we used a 4-digit LCD from Ocular Inc. [1]. The circuit diagram used in this application report is shown in Figure 6. Each I/O pin on the PIC16C55 device controls the state of two segments (see Figure 6) which requires a total of 16 I/O pins. The reference voltages are generated through a simple to input, which tristates the voltage level seen on the pin. This method uses 4 I/O pins to generate the proper voltage levels. Figure 7 shows the truth table for generating the voltage levels. Figure 8 shows how to create a bitmap for different digits. Figure 9 shows the waveforms generated for the accompanying software which implements a hexadecimal counter.

FIGURE 6: SYSTEM CONFIGURATION WITH LCD PINOUT

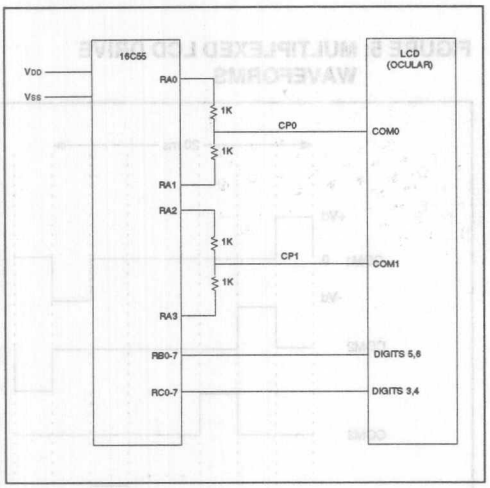


FIGURE 7: COMMON PLANE SIGNAL GENERATION

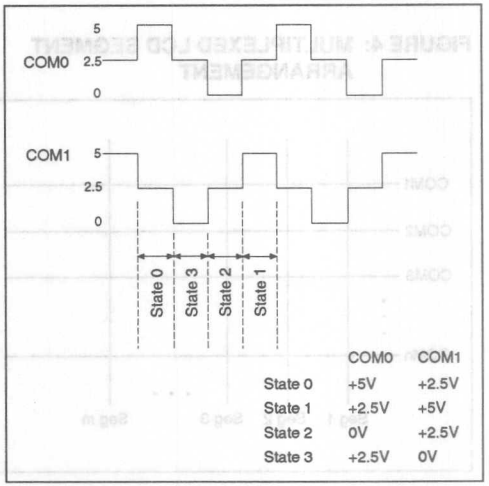
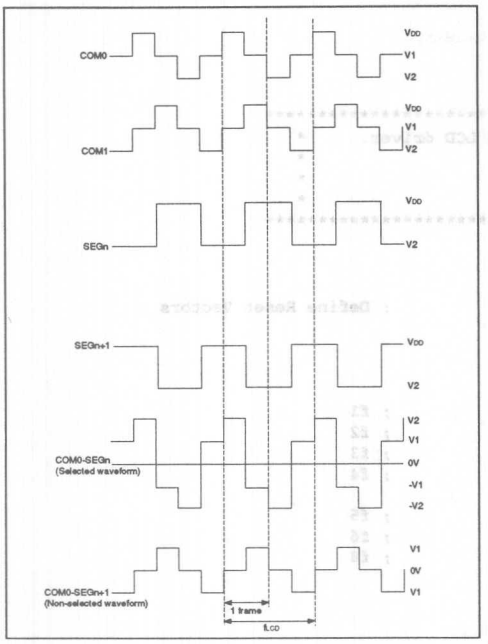


FIGURE 8: LCD CHARACTER BITMAP

Digit	COM0 SEG 0				COM1 SEG 1				COM0 SEG 2				COM1 SEG3			
	F	E	D	DP	A	G	C	B	F	E	D	DP	A	G	C	B
0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
3	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1
4	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1
5	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1
6	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
7	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1
8	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1
9	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
a	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1
b	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
c	1	0	0	1	1	1	0	1	0	1	1	0	0	0	1	0
d	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1
e	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0
f	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

© 1993 Microchip Technology Inc.

FIGURE 9: EXAMPLE OF OUTPUT WAVEFORMS FOR DIGIT 4



CONCLUSION

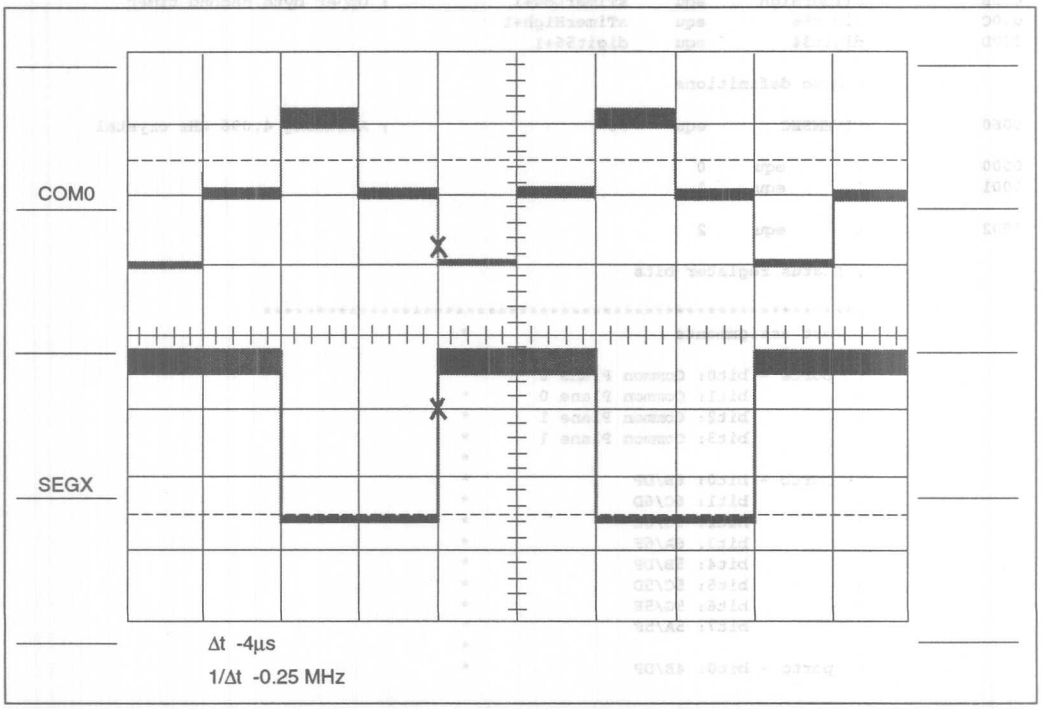
In this application report we have demonstrated the use of PIC16C5X devices to implement a simple LCD controller. As discussed earlier, it is important to keep the generated DC voltage to a minimum to extend the life of the LCD. Ideally one should switch all the I/O lines simultaneously, however, a software implementation of the LCD controller will necessarily introduce a delay which is proportional to the instruction cycle of the microcontroller, as shown in Figure 10. Therefore it is necessary to keep the switching time to a minimum. Our implementation introduced less than 50 mV of DC voltage on the segment lines which is below the manufacturer's recommended DC offset voltage of 60 mV.

REFERENCES

[1] Ocular Inc., Drawing number JH074.

AUTHOR: Al Lovrich
Logic Products Division

FIGURE 10: MICROCONTROLLER GENERATED OUTPUT WAVEFORM



© 1993 Microchip Technology Inc.

```
MPASM B0.54                                                    PAGE  1

        LIST    C=132,n=0,p=16c55,r=dec

        ;*********************************************************
        ; Project: PIC16C5X as a multiplexed LCD driver.        *
        ;                                                      *
        ; Revision history:                                    *
        ;       04/14/93        original                       *
        ;*********************************************************

        ; Equates

01FF    pic54   equ     0x1ff                   ; Define Reset Vectors
01FF    pic55   equ     0x1ff
03FF    pic56   equ     0x3ff
07FF    pic57   equ     0x7ff

0001    rtcc    equ     1                       ; f1
0002    pc      equ     2                       ; f2
0003    status  equ     3                       ; f3
0004    fsr     equ     4                       ; f4

0005    porta   equ     5                       ; f5
0006    portb   equ     6                       ; f6
0007    portc   equ     7                       ; f8

        ; realtime mode registers

0008    currentState    equ     8
0009    msTimer         equ     currentState+1  ; Millisecond timer
000A    sTimerLow       equ     msTimer+1       ; Lower byte second timer
000B    sTimerHigh      equ     sTimerLow+1     ; Upper byte second timer
000C    digit56         equ     sTimerHigh+1
000D    digit34         equ     digit56+1

        ; Misc definitions

0060    FIVEMSEC        equ     96              ; Assuming 4.096 MHz crystal

0000    w       equ     0
0001    f       equ     1

0002    z       equ     2

        ; Status register bits

        ;*********************************************************
        ; Port assignments                                     *
        ;                                                      *
        ;  porta - bit0: Common Plane 0                        *
        ;          bit1: Common Plane 0                        *
        ;          bit2: Common Plane 1                        *
        ;          bit3: Common Plane 1                        *
        ;                                                      *
        ;  portb - bit0: 6B/DP                                 *
        ;          bit1: 6C/6D                                 *
        ;          bit2: 6G/6E                                 *
        ;          bit3: 6A/6F                                 *
        ;          bit4: 5B/DP                                 *
        ;          bit5: 5C/5D                                 *
        ;          bit6: 5G/5E                                 *
        ;          bit7: 5A/5F                                 *
        ;                                                      *
        ;  portc - bit0: 4B/DP                                 *
```


© 1993 Microchip Technology Inc.

```
;       - bit1: 4C/4D          *
;       - bit2: 4G/4E          *
;       - bit3: 4A/4F          *
;       - bit4: 3B/DP          *
;       - bit5: 3C/3D          *
;       - bit6: 3G/3E          *
;       - bit7: 3A/3F          *
;                              *
;*****************************************************

;*****************************************************
;       Macro definitions      *
;*****************************************************
UpdateState     macro   State, Table

        swapf   sTimerLow, w
        andlw   0xf                     ; Isolate digit 5 (offset)
        call    Table
        movwf   digit56
        swapf   digit56, f

        movf    sTimerLow, w
        andlw    0xf                    ; Isolate digit 6 (offset)
        call    Table
        iorwf   digit56, f

        swapf   sTimerHigh, w
        andlw   0xf                     ; Isolate digit 5 (offset)
        call    Table
        movwf   digit34
        swapf   digit34, f

        movf    sTimerHigh, w
        andlw   0xf                     ; Isolate digit 6 (offset)
        call    Table
        iorwf   digit34, f

        movf    digit34, w              ; Display digits 3 & 4
        movwf   portc
        movf    digit56, w
        movwf   portb                   ; Display digits 5 & 6
        endm

        org     0

; Initialize ports A, B, and C and RTCC. In case of output data
; values, set the data latch first, then set the port direction.

Initialize
0000 0C01       movlw   00000001b               ; Set data latch
0001 0025       movwf   porta
0002 0C08       movlw   00001000b               ; Set I/O direction
0003 0005       tris    porta

0004 0C00       movlw   00000000b               ; Set levels to low
0005 0026       movwf   portb
0006 0C00       movlw   00000000b               ; Set as outputs
0007 0006       tris    portb

0008 0C00       movlw   00000000b               ; Set levels to low
0009 0027       movwf   portc
000A 0C00       movlw   00000000b               ; Set as outputs
000B 0007       tris    portc

000C 0C04       movlw   0x04                    ; Set prescaler
000D 0002       option

000E 0C60       movlw   FIVEMSEC                ; rtcc = 5ms
000F 0021       movwf   rtcc
```

© 1993 Microchip Technology Inc.

```
0011 0020           movwf   currentState

0012 0C0D           movlw   0xd
0013 0029           movwf   msTimer                 ; Initialize millisecond timer

0014 006A           clrf    sTimerLow               ; Clear second counter
0015 006B           clrf    sTimerHigh

0016 0800           retlw   0

                ; Check timer register for timing out (rtcc = 0). Remain in the
                ; loop until the timer times out.


                ; Wait for 5ms timer timeout
                Timer_Check
0017 0201           movf    rtcc, w
0018 0743           btfss   status, z
0019 0A17           goto    Timer_Check

001A 0C60           movlw   FIVEMSEC
001B 0021           movwf   rtcc

001C 02E9           decfsz  msTimer, f
001D 0A21           goto    Update_Backplane

001E 03EA           incfsz  sTimerLow               ; Update second counter
001F 0A21           goto    Update_Backplane
0020 02AB           incf    sTimerHigh, f

                ; RA0 and RA1 are used to control voltage level for common plane 0.
                ; RA2 and RA3 are used to control voltage level for common plane 1.
                ; There are four possible states with different voltage levels as
                ; follows:
                ;
                ; State 0 - cp0 = +5v     ra0=1,  ra1=x
                ;           cp1 = +2.5v   ra2=1,  ra3=0
                ; State 1 - cp0 = +2.5v   ra0=1,  ra1=0
                ;           cp1 = +5v     ra2=1,  ra3=x
                ; State 2 - cp0 =  0v     ra0=0,  ra1=x
                ;           cp1 = +2.5v   ra2=1,  ra3=0
                ; State 3 - cp0 = +2.5v   ra0=1,  ra1=0
                ;           cp1 =  0v     ra2=0,  ra3=x

                Update_Backplane
0021 0004           clrwdt                          ; Reset watchdog timer

0022 00C8           decf    currentState, w         ; Update w register
0023 0E03           andlw   0x03                    ; Use only bit0/1
0024 0028           movwf   currentState            ; Update currentState
0025 01E2           addwf   pc, f

0026 0AAD           goto    State3
0027 0A81           goto    State2
0028 0A55           goto    State1
                ;   goto    State0

                ;           State 0
                            State0
                UpdateState State0, S0_Table

0029 038A           swapf   sTimerLow, w
002A 0E0F           andlw   0xf                 ; Isolate digit 5 (offset)
002B 0944           call    S0_Table
```


© 1993 Microchip Technology Inc.

```
002C 002C           movwf   digit56
002D 03AC           swapf   digit56, f

002E 020A           movf    sTimerLow, w
002F 0E0F           andlw   0xf                 ; Isolate digit 6 (offset)
0030 0944           call    S0_Table
0031 012C           iorwf   digit56, f

0032 038B           swapf   sTimerHigh, w
0033 0E0F           andlw   0xf                 ; Isolate digit 5 (offset)
0034 0944           call    S0_Table
0035 002D           movwf   digit34
0036 03AD           swapf   digit34, f

0037 020B           movf    sTimerHigh, w
0038 0E0F           andlw   0xf                 ; Isolate digit 6 (offset)
0039 0944           call    S0_Table
003A 012D           iorwf   digit34, f

003B 020D           movf    digit34, w          ; Display digits 3 & 4
003C 0027           movwf   portc
003D 020C           movf    digit56, w
003E 0026           movwf   portb               ; Display digits 5 & 6

003F 0C05           movlw   00000101b
0040 0025           movwf   porta
0041 0C02           movlw   00000010b
0042 0005           tris    porta

0043 0800           retlw   0

                S0_Table
0044 01E2           addwf    pc, f              ; Add offset to pc

0045 0804           retlw   0100b               ; 0
0046 080C           retlw   1100b               ; 1
0047 0802           retlw   0010b               ; 2
0048 0800           retlw   0000b               ; 3
0049 0808           retlw   1000b               ; 4
004A 0801           retlw   0001b               ; 5
004B 080F           retlw   1111b               ; 6
004C 0804           retlw   0100b               ; 7
004D 0800           retlw   0000b               ; 8
004E 0800           retlw   0000b               ; 9
004F 0800           retlw   0000b               ; a
0050 0809           retlw   1001b               ; b
0051 080B           retlw   1011b               ; c
0052 0808           retlw   1000b               ; d
0053 0803           retlw   0011b               ; e
0054 0803           retlw   0011b               ; f

                ; State 1

                State1
                            UpdateState     State1, S1_Table

0055 038A           swapf   sTimerLow, w
0056 0E0F           andlw   0xf                 ; Isolate digit 5 (offset)
0057 0970           call    S1_Table
0058 002C           movwf   digit56
0059 03AC           swapf   digit56, f

005A 020A           movf    sTimerLow, w
005B 0E0F           andlw   0xf                 ; Isolate digit 6 (offset)
005C 0970           call    S1_Table
005D 012C           iorwf   digit56, f

005E 038B           swapf   sTimerHigh, w
```

© 1993 Microchip Technology Inc.

```
005F 0E0F             andlw   0xf                    ; Isolate digit 5 (offset)
0060 0970             call    S1_Table
0061 002D             movwf   digit34
0062 03AD             swapf   digit34, f

0063 020B             movf    sTimerHigh, w
0064 0E0F             andlw   0xf                    ; Isolate digit 6 (offset)
0065 0970             call    S1_Table
0066 012D             iorwf   digit34, f

0067 020D             movf    digit34, w             ; Display digits 3 & 4
0068 0027             movwf   portc
0069 020C             movf    digit56, w
006A 0026             movwf   portb                  ; Display digits 5 & 6



006B 0C05             movlw   00000101b
006C 0025             movwf   porta
006D 0C08             movlw   00001000b
006E 0005             tris    porta

006F 0800             retlw   0

                    S1_Table
0070 01E2             addwf    pc, f

0071 0801             retlw   0001b                  ; 0
0072 080F             retlw   1111b                  ; 1
0073 0809             retlw   1001b                  ; 2
0074 080D             retlw   1101b                  ; 3
0075 0807             retlw   0111b                  ; 4
0076 0805             retlw   0101b                  ; 5
0077 080F             retlw   1111b                  ; 6
0078 080F             retlw   1111b                  ; 7
0079 0801             retlw   0001b                  ; 8
007A 0805             retlw   0101b                  ; 9
007B 0803             retlw   0011b                  ; a
007C 0801             retlw   0001b                  ; b
007D 0809             retlw   1001b                  ; c
007E 0809             retlw   1001b                  ; d
007F 0801             retlw   0001b                  ; e
0080 0803             retlw   0011b                  ; f

                    ; State 2

                    State2
                    UpdateState     State2, S2_Table

0081 038A             swapf   sTimerLow, w
0082 0E0F             andlw   0xf                    ; Isolate digit 5 (offset)
0083 099C             call    S2_Table
0084 002C             movwf   digit56
0085 03AC             swapf   digit56, f

0086 020A             movf    sTimerLow, w
0087 0E0F             andlw   0xf                    ; Isolate digit 6 (offset)
0088 099C             call    S2_Table
0089 012C             iorwf   digit56, f

008A 038B             swapf   sTimerHigh, w
008B 0E0F             andlw   0xf                    ; Isolate digit 5 (offset)
008C 099C             call    S2_Table
008D 002D             movwf   digit34
008E 03AD             swapf   digit34, f

008F 020B             movf    sTimerHigh, w
0090 0E0F             andlw   0xf                    ; Isolate digit 6 (offset)
0091 099C             call    S2_Table
```


© 1993 Microchip Technology Inc.

```
0092 012D            iorwf   digit34, f

0093 020D            movf    digit34, w          ; Display digits 3 & 4
0094 0027            movwf    portc
0095 020C            movf    digit56, w
0096 0026            movwf   portb               ; Display digits 5 & 6


0097 0C04            movlw   00000100b
0098 0025            movwf   porta
0099 0C02            movlw   00000010b
009A 0005            tris    porta

009B 0800            retlw   0

                S2_Table
009C 01E2            addwf   pc, f

009D 080B            retlw   1011b   ; 0
009E 0803            retlw   0011b   ; 1
009F 080D            retlw   1101b   ; 2
00A0 080F            retlw   1111b   ; 3
00A1 0807            retlw   0111b   ; 4
00A2 080E            retlw   1110b   ; 5
00A3 080E            retlw   1110b   ; 6
00A4 080B            retlw   1011b   ; 7
00A5 080F            retlw   1111b   ; 8
00A6 080F            retlw   1111b   ; 9
00A7 080F            retlw   1111b   ; a
00A8 0806            retlw   0110b   ; b
00A9 0804            retlw   0100b   ; c
00AA 0807            retlw   0111b   ; d
00AB 080C            retlw   1100b   ; e
00AC 080C            retlw   1100b   ; f

                ; State 3

                State3
                UpdateState     State3, S3_Table

00AD 038A            swapf   sTimerLow, w
00AE 0E0F            andlw   0xf                 ; Isolate digit 5 (offset)
00AF 09C8            call    S3_Table
00B0 002C            movwf   digit56
00B1 03AC            swapf   digit56, f

00B2 020A            movf    sTimerLow, w
00B3 0E0F            andlw   0xf                 ; Isolate digit 6 (offset)
00B4 09C8            call    S3_Table
00B5 012C            iorwf   digit56, f

00B6 038B            swapf   sTimerHigh, w
00B7 0E0F            andlw   0xf                 ; Isolate digit 5 (offset)
00B8 09C8            call    S3_Table
00B9 002D            movwf   digit34
00BA 03AD            swapf   digit34, f

00BB 020B            movf    sTimerHigh, w
00BC 0E0F            andlw   0xf                 ; Isolate digit 6 (offset)
00BD 09C8            call    S3_Table
00BE 012D            iorwf   digit34, f

00BF 020D            movf    digit34, w          ; Display digits 3 & 4
00C0 0027            movwf   portc
00C1 020C            movf    digit56, w
00C2 0026            movwf   portb          ; Display digits 5 & 6
```

© 1993 Microchip Technology Inc.

```
00C6 0005               tris    porta

00C7 0800               retlw   0

                        S3_Table
00C8 01E2               addwf   pc, f

00C9 080E               retlw   1110b           ; 0
00CA 0800               retlw   0000b           ; 1
00CB 0806               retlw   0110b           ; 2
00CC 0802               retlw   0010b           ; 3
00CD 0808               retlw   1000b           ; 4
00CE 080A               retlw   1010b           ; 5
00CF 080E               retlw   1110b           ; 6
00D0 0800               retlw   0000b           ; 7
00D1 080E               retlw   1110b           ; 8
00D2 080A               retlw   1010b           ; 9
00D3 080C               retlw   1100b           ; a
00D4 080E               retlw   1110b           ; b
00D5 0806               retlw   0110b           ; c
00D6 0806               retlw   0110b           ; d
00D7 080E               retlw   1110b           ; e
00D8 080C               retlw   1100b           ; f

                        ; Main code

                        Start
00D9 0900                       call    Initialize
                        Repeat
00DA 0917                       call    Timer_Check
00DB 0ADA                       goto    Repeat

                                org     pic55

                        System_Reset
01FF 0AD9                               goto    Start

                            END

Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

PLD Replacement

INTRODUCTION

The PIC16C5X microcontrollers are ideal for implementing low cost combinational and sequential logic circuits that traditionally have been implemented using either numerous TTL gates or using programmable logic chips such as PLAs or EPLDs.

PIC16C5X is a family of high-performance 8-bit microcontrollers from Microchip Technology. It employs Harvard architecture, i.e has a separate data bus (8-bit) and a program bus (12-bit wide). All instructions are single word and execute in one cycle except for program branches. The instruction cycle time is 200 ns at 20 MHz and faster versions with clock frequency of 20 MHz (instruction cycle = 200 ns) are planned. The PIC16C5X microcontrollers are ideal for PLD-type application because:

* Very low cost. Extremely cost effective to replace TTL gates or expensive EPLD's.
* Fully programmable. PIC16C5X microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* PC board real estate saving can be substantial when replacing multitude of TTL's or several PLD's with PIC16C5X microcontrollers which are packaged in 18 and 28 pin packages (DIP, PLCC, SOIC).
* Substantial power savings can be attained by using PIC16C5X's SLEEP mode. In this mode typical power consumption of PIC16C5X is less than 1uA.
* PIC16C5X's I/O ports are bidirectional and software configurable as input or output. The user can mix and match number of inputs or outputs as long as the total does not exceed 20 (PIC16C55/57).
* PIC16C5X's output pins have large current source/sink capability. They can directly drive LED's.
* The speed and efficiency of the PIC16C5X allows it to perform other control, timing, and compute functions in addition to implementing a PLA function.

IMPLEMENTING A PLA

To implement a generic combinational logic function, we can simply emulate an AND-OR PLA in software. This will require that the logic outputs be described as sum of products. To describe our algorithm, we will use a simple 8-input, 8-bit output PLA with 24 product terms (Figure 1). We will further use the truth table in Figure 2 as the PLA function being implemented. In this example, only four inputs (A3: A0) are used and the other four inputs (A7: A4) are don't care. On the output side, seven output pins (Y6: Y0) are used and Y7 is unused. To implement this PLA, the logic inputs A0,A1,...,A7 can be connected to port RB pins RB0,RB1,...,RB7 respectively. The logic outputs Y0,Y1,...,Y7 will appear on port RC pins RC0,RC1,...,RC7 respectively. Port RB is configured as input and port RC will be configured as output. To evaluate each product term one XOR (exclusive OR) and one AND operation will be required. For example, to determine product term P3 = A3.A2.A1.A0 , the expression to evaluate is:

(A<7:0> .XOR. XXXX0011B) .AND. 00001111B).

The constant with which XOR is done will be referred to as P3_x in our discussion. P3_x = XXXX0011B will ensure that if A<3:0> = 0011B, the least significant 4 bits of the result will be 0000b. The AND constant, referred to here as P3_a (Product term 3, AND constant) basically eliminates the don't care inputs (here A<7:4>) by masking them. Therefore, if the result of the XOR-AND operation is zero then P3 = 1 else P3 = 0. Once the Product terms are evaluated they are stored in four product registers Preg_0 to Preg_3. To determine an output term:

Y0 = P0 + P2 + P3 + P5 + P6 + P7 + P8 + P9 + P10 + P12 + P13 + P14

we need to evaluate the following expression:

(Preg_a .AND. OR_a0) .OR. (Preg_b .AND. OR_b0) .OR. (Preg_c .AND. OR_c0)

In our case the constant values to implement Y0 are as follows:

OR_a0 =	1	1	1	0	1	1	0	1
	P7	P6	P5		P3	P2		P0

OR_b0 = 11010111

OR_c0 = 00000000

For larger number of inputs, outputs or product terms, the evaluation will be more complex but following the same principle. Appendix A shows the assembly code to implement this 8 input X 8 output X 24 Product PLA. This example optimizes speed as well as program memory requirement. Appendix B shows a slightly different implementation (only EVAL_Y MACRO is different) that optimizes program memory usage over speed. Table 1 shows time and resources required to implement different size PLAs.

© 1994 Microchip Technology Inc.

FIGURE 1 - A SIMPLE PLA

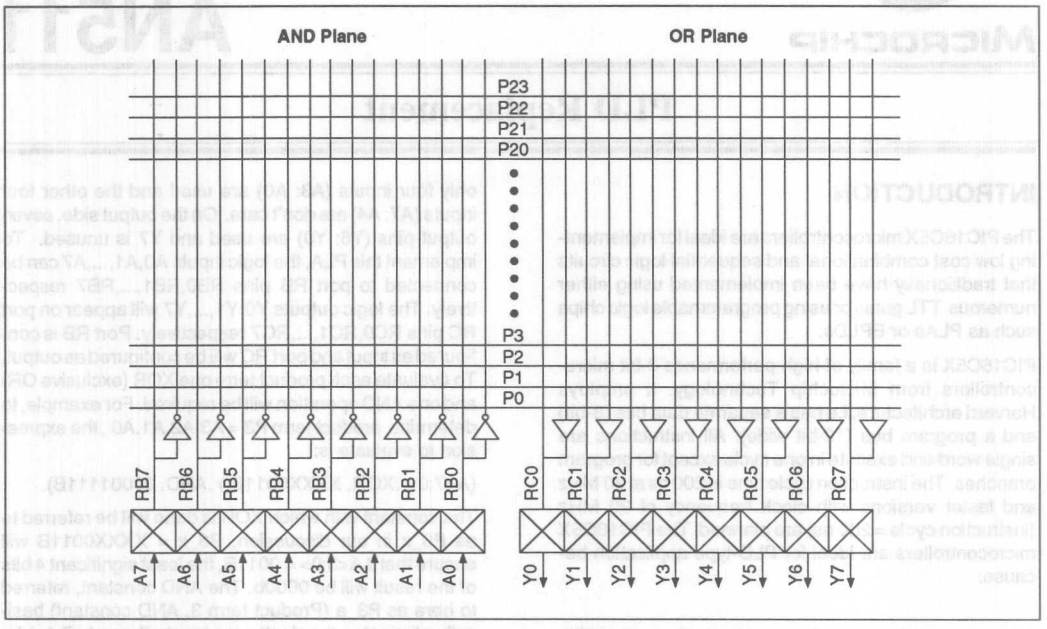
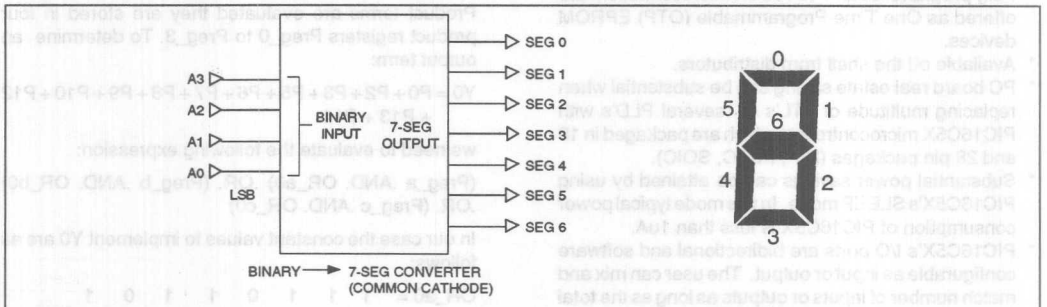


FIGURE 2 - BINARY TO 7-SEGMENT CONVERSION EXAMPLE



Truth Table:

HEX	A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Product Terms
0	0	0	0	0	1	1	1	1	1	1	0	P0 = $\overline{A3}$, $\overline{A2}$, $\overline{A1}$, $\overline{A0}$
1	0	0	0	1	0	1	1	0	0	0	0	P1
2	0	0	1	0	1	1	0	1	1	0	1	P2
3	0	1	1	1	1	1	1	1	0	0	1	P3
4	0	1	0	0	0	1	1	0	0	1	1	P4
5	0	1	0	1	1	0	1	1	0	1	1	P5
6	0	1	1	0	1	0	1	1	1	1	1	P6
7	0	1	1	1	1	1	1	0	0	0	0	P7
8	1	0	0	0	1	1	1	1	1	1	1	P8
9	1	0	0	1	1	1	1	0	0	1	1	P9
A	1	0	1	0	1	1	1	0	1	1	1	P10
B	1	0	1	1	0	0	0	1	1	1	1	P11
C	1	1	0	0	1	0	0	1	1	1	0	P12
D	1	1	0	1	0	1	1	1	1	0	1	P13
E	1	1	1	0	1	0	0	1	1	1	1	P14
F	1	1	1	1	1	0	0	0	1	1	1	P15 = A3, A2, A1, A0

© 1994 Microchip Technology Inc.

SPEED/RESPONSE TIME

The worst case response time of a PLA implemented in this fashion can be calculated as follows. First, we define td = time required to execute the PLA program assuming the worst case program branches are taken. Then the maximum propagation delay time from input change to valid output = 2td. This is because if an input changes just after the program reads input port, its effect will not show up until the program completes the current execution cycle, re-reads input and recalculates output. This is shown in Figure 3. There are ways to improve the delay time such as sample inputs several times throughout the PLA program and if input change is sensed, return to the beginning rather than execute the rest of the evaluation code.

A Table Look-Up Method For Small PLA Implementation

If the number of inputs is small (8 or less) then a simple table look-up method can be used to implement the PLA. This will improve execution time to around 5 µs (@ 8 MHz input clock). The following code implements the BCD to 7-segment conversion (Figure 2) using this technique.

```
begin   movlw   0ffh        ;
        tris    6           ;Port_b = input
        clrw                ;
        tris    7           ;Port_c = output
pla 88  movf    Port_b, w   ;Read input
        andlw   0fh         ;Mask off bits 7:4
        call    op_tbl      ;
        movwf   Port_c      ;Write output
        goto    pla88       ;
op_tbl  addwf   pc          ;Computed jump for
                             table look-up
        retlw   b"00111111" ;
        retlw   b"00000110" ;
        retlw   b"01011011" ;
        retlw   b"01001111" ;
        retlw   b"01100110" ;
        retlw   b"01101101" ;
        retlw   b"01111101" ;
        retlw   b"00000111" ;
        retlw   b"01111111" ;
        retlw   b"01100111" ;
        retlw   b"01110111" ;
        retlw   b"01111000" ;
        retlw   b"00111001" ;
        retlw   b"01011110" ;
        retlw   b"01111001" ;
        retlw   b"01110001" ;
```

TABLE 1 - EXECUTION TIME AND RESOURCES NECESSARY FOR DIFFERENT SIZE PLA's

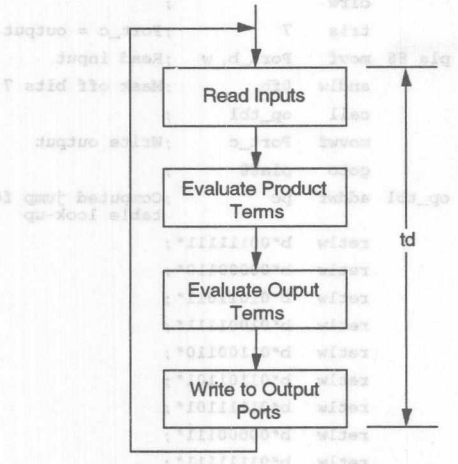
Number of Inputs Including fdbk	Number of Outputs Including fdbk and o/e Control	Number of Products	Number of RAM Locations Required NRAM	Number of Program Memory Locations Required NROM	Number of Instruction Cycle To Execute PLA NCYC	Real Time @ 20 MHz to Execute PLA	
8	8	24	5	228	228	45.6 µs	Time Efficient
			10	222	352	70.4 µs	Code Efficient
8	8	48	8	447	447	89.4 µs	Time Efficient
			13	384	535	107 µs	Code Efficient
16	16	64	12	1042	1042	208.4 µs	Time Efficient
			22	843	1250	250 µs	Code Efficient
20	24	80	16	1661	1661	372.2 µs	Time Efficient
			40	1462	2221	444.2 µs	Code Efficient

If Ni = Number of inputs
NIW = Number of input words, NIW = $\left[\frac{Ni}{8}\right]$
NP = Number of products
NPW = Number of product words, i.e. NPW = $\left[\frac{Np}{8}\right]$
NO = Number of outputs
NOW = Number of output words, i.e. NOW = $\left[\frac{No}{8}\right]$

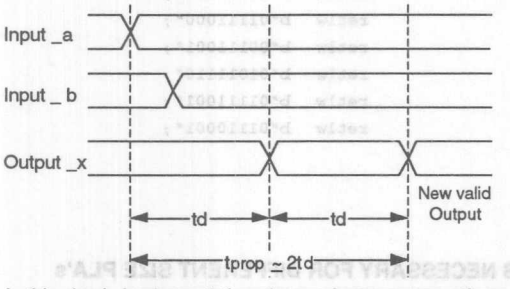
Then,
NRAM @ NIW + NOW + N PW : Time efficient
NRAM @ NIW + NOW + 2 N PW + 2: Code efficient
NROM @ 8 + N PW + NOW + N P [2 + 3 NIW] +NO • NPW • 4: Time efficient
NROM @ 17 + N PW + NOW + N P [2 + 3 NIW] +NO [NPW + 3]: Code efficient
NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] +NO [2 NPW • 4]: Time efficient
NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] +5 N [NPW + 1]: Code efficient

© 1994 Microchip Technology Inc.

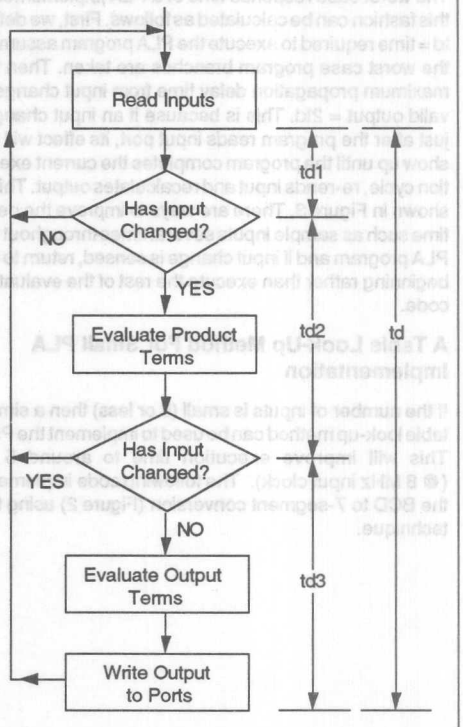
FIGURE 3 - PLA PROGRAM FLOW



A very simple PLA evaluation flow.



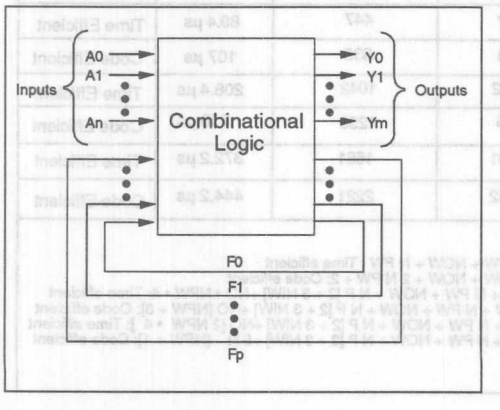
In this simple implementation, the maximum propragation delay from input change to output change is ~ 2td



More complex program flow can be used, such as this one to reduce tprop.

In this example td < t prop < td + td3

FIGURE 4 - AN ASYNCHRONOUS STATE MACHINE

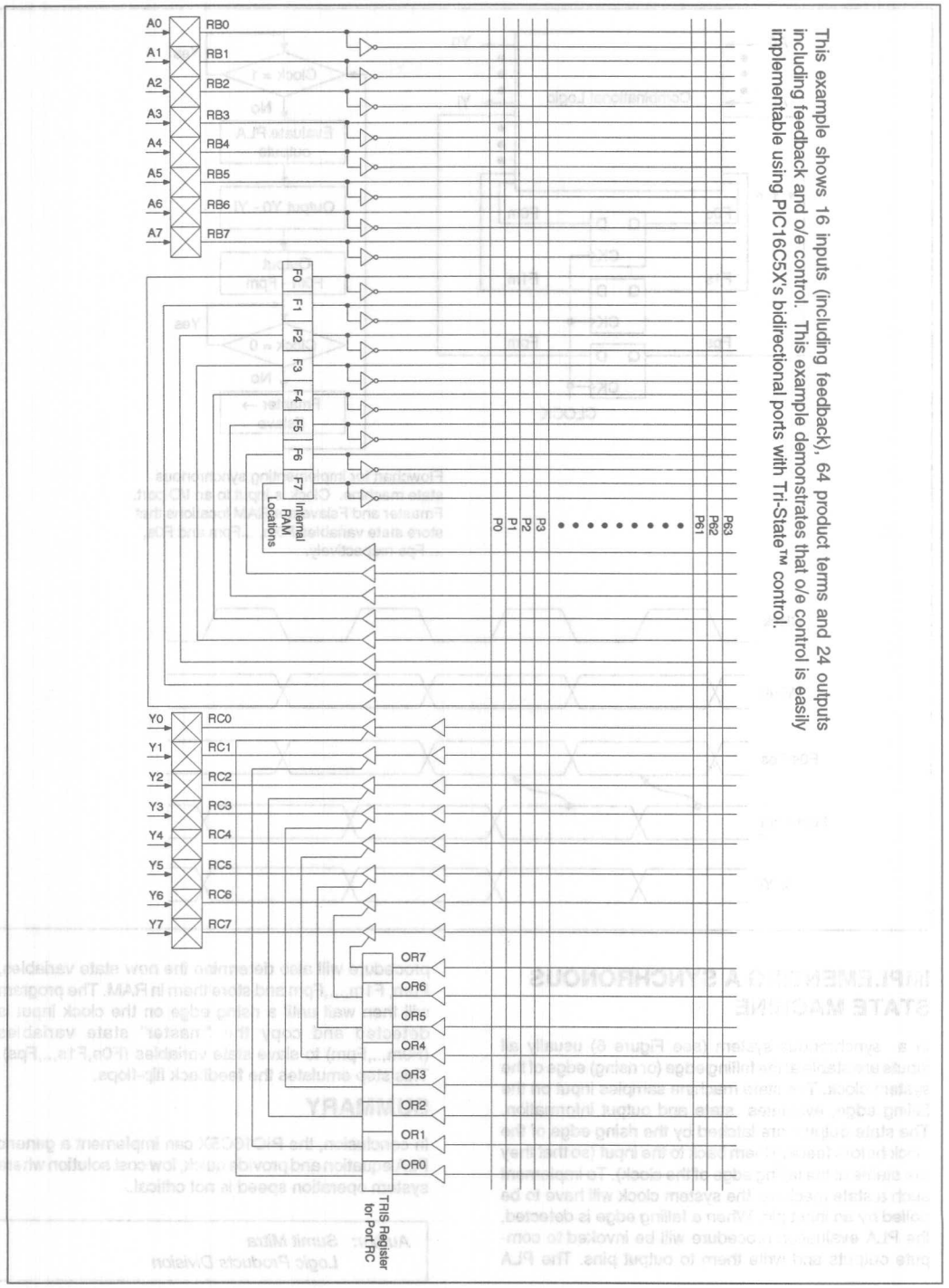


IMPLEMENTING AN ASYNCHRONOUS STATE MACHINE

The concept can be easily extended to implement sequential logic i.e. a state machine. Figure 4 shows a state machine with n inputs (A0-An), m outputs (Y0-Ym) and p states that feedback as inputs to the PLA (F0-Fp). In PIC16C5X the states will be stored as bits in RAM location. Input will now mean input from a port as well as from the feedback registers. Figure 5 shows an example PLA with eight inputs A0,A1,..,A7 that are connected to port RB pins RB0,RB1,..,RB7. This example PLA has a total of 24 outputs of which eight are actual outputs, another eight are output enable control for the outputs and the other eight are feedbacks (or states). The PLA shown here, therefore, in essence implements an asynchronous state machine (i.e. there is no system clock).

© 1994 Microchip Technology Inc.

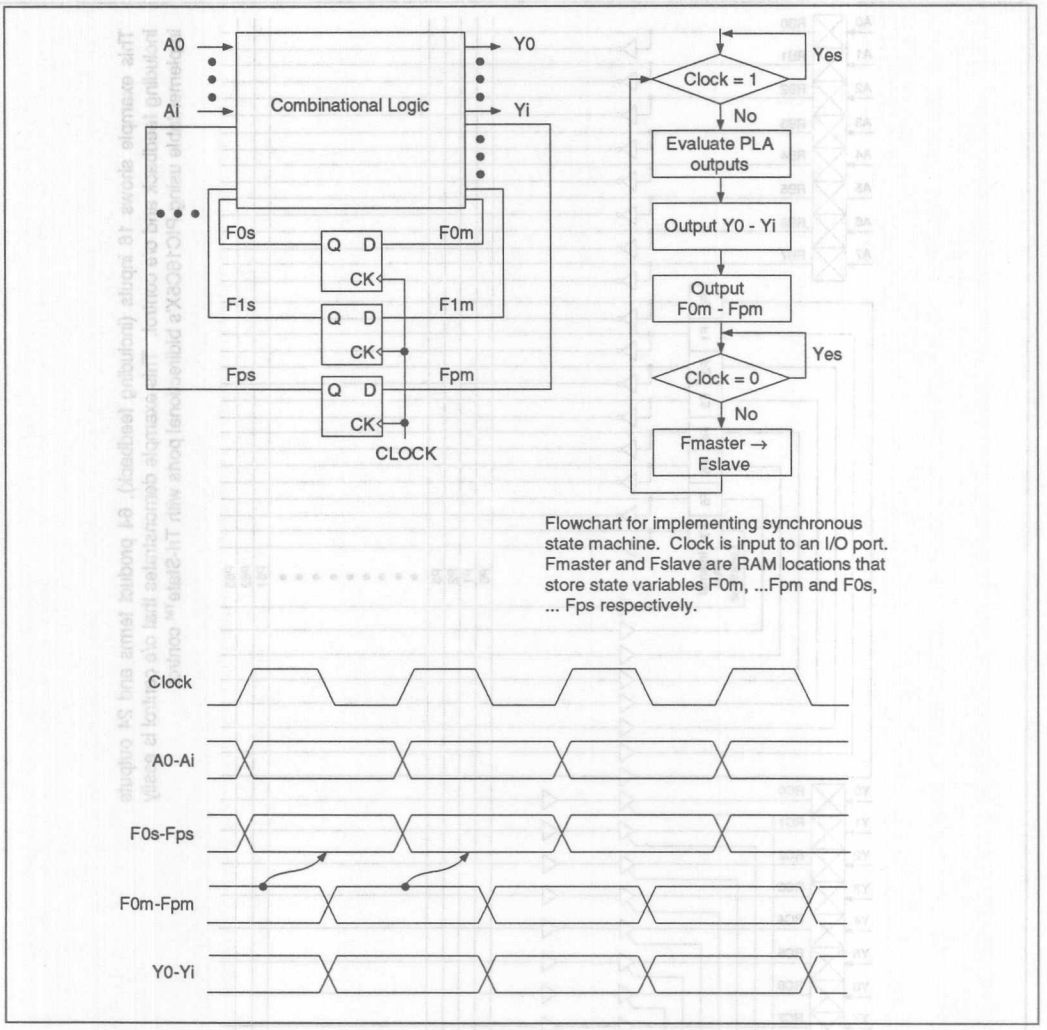
FIGURE 5 - EXAMPLE OF A LARGER PLA IMPLEMENTATION



This example shows 16 inputs (including feedback), 64 product terms and 24 outputs including feedback and o/e control. This example demonstrates that o/e control is easily implementable using PIC16C5X's bidirectional ports with Tri-State™ control.

© 1994 Microchip Technology Inc.

FIGURE 6 - A SYNCHRONOUS STATE MACHINE IMPLEMENTATION



Flowchart for implementing synchronous state machine. Clock is input to an I/O port. Fmaster and Fslave are RAM locations that store state variables F0m, ...Fpm and F0s, ... Fps respectively.

IMPLEMENTING A SYNCHRONOUS STATE MACHINE

In a synchronous system (see Figure 6) usually all inputs are stable at the falling edge (or rising) edge of the system clock. The state machine samples input on the falling edge, evaluates state and output information. The state outputs are latched by the rising edge of the clock before feeding them back to the input (so that they are stable at the falling edge of the clock). To implement such a state machine, the system clock will have to be polled by an input pin. When a falling edge is detected, the PLA evaluation procedure will be invoked to compute outputs and write them to output pins. The PLA procedure will also determine the new state variables, F0m, F1m,...,Fpm and store them in RAM. The program will then wait until a rising edge on the clock input is detected and copy the "master" state variables (F0m,..,Fpm) to slave state variables (F0s,F1s,..,Fps). This step emulates the feedback flip-flops.

SUMMARY

In conclusion, the PIC16C5X can implement a generic PLA equation and provide quick, low cost solution where system operation speed is not critical.

Author: Sumit Mitra
Logic Products Division

© 1994 Microchip Technology Inc.

APPENDIX A: PLA IMPLEMENTATION: TIME EFFICIENT APPROACH

```
MPASM B0.54                                                    PAGE  1


                        ;*****************************************************************
                        ; pla1a.asm :
                        ; This procedure implements a simple AND-OR PLA with:
                        ;
                        ;       8  inputs         := A7 A6 A5 A4 A3 A2 A1 A0
                        ;       24 product terms  := P23 P22 ..... P0
                        ;       8  outputs        := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                        ;
                        ; The eight inputs are assumed to be connected to PORT RB such that
                        ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                        ; The outputs are programmed to appear on port RC such that
                        ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                        ;
                        ; This implementation optimizes both speed & program memory usage
                        ;
                        ;*****************************************************************
                        ;
                        ; define RAM locations used:
                        ;
                                LIST    P=16C57
000C                    input   equ     d'12'           ;RAM location 12 holds input
000D                    Y_reg   equ     d'13'           ;holds output result

000E                    Preg_a  equ     d'14'           ;Product terms P0 to P7. Preg_a<0> = P0
000F                    Preg_b  equ     d'15'           ;Product terms P8 to P15. Preg_b<0> = P8
0010                    Preg_c  equ     d'16'           ;Product terms P16 to P23. Preg_c<0> =
P16
                        ; define some constants and file addresses:
                        ;
0000                    bit0    equ     0               ;
0001                    bit1    equ     1               ;
0002                    bit2    equ     2               ;
0003                    bit3    equ     3               ;
0004                    bit4    equ     4               ;
0005                    bit5    equ     5               ;
0006                    bit6    equ     6               ;
0007                    bit7    equ     7               ;
                        ;
0003                    status  equ     3               ;
0006                    port_b  equ     6               ;
0007                    port_c  equ     7               ;
                        ;
                        ; define the AND plane programming variables:
                        ;
0000                    P0_x    equ     b'00000000'     ;
000F                    P0_a    equ     b'00001111'     ;
0001                    P1_x    equ     b'00000001'     ;
000F                    P1_a    equ     b'00001111'     ;
0002                    P2_x    equ     b'00000010'     ;
000F                    P2_a    equ     b'00001111'     ;
0003                    P3_x    equ     b'00000011'     ;
000F                    P3_a    equ     b'00001111'     ;
0004                    P4_x    equ     b'00000100'     ;
000F                    P4_a    equ     b'00001111'     ;
0005                    P5_x    equ     b'00000101'     ;
000F                    P5_a    equ     b'00001111'     ;
0006                    P6_x    equ     b'00000110'     ;
000F                    P6_a    equ     b'00001111'     ;
0007                    P7_x    equ     b'00000111'     ;
000F                    P7_a    equ     b'00001111'     ;
0008                    P8_x    equ     b'00001000'     ;
000F                    P8_a    equ     b'00001111'     ;
0009                    P9_x    equ     b'00001001'     ;
```

```
000F                P9_a            equ     b'00001111'     ;
000A                P10_x           equ     b'00001010'     ;
000F                P10_a           equ     b'00001111'     ;
000B                P11_x           equ     b'00001011'     ;
000F                P11_a           equ     b'00001111'     ;
000C                P12_x           equ     b'00001100'     ;
000F                P12_a           equ     b'00001111'     ;
000D                P13_x           equ     b'00001101'     ;
000F                P13_a           equ     b'00001111'     ;
000E                P14_x           equ     b'00001110'     ;
000F                P14_a           equ     b'00001111'     ;
000F                P15_x           equ     b'00001111'     ;
000F                P15_a           equ     b'00001111'     ;
0000                P16_x           equ     b'00000000'     ;
0000                P16_a           equ     b'00000000'     ;
0000                P17_x           equ     b'00000000'     ;
0000                P17_a           equ     b'00000000'     ;
0000                P18_x           equ     b'00000000'     ;
0000                P18_a           equ     b'00000000'     ;
0000                P19_x           equ     b'00000000'     ;
0000                P19_a           equ     b'00000000'     ;
0000                P20_x           equ     b'00000000'     ;
0000                P20_a           equ     b'00000000'     ;
0000                P21_x           equ     b'00000000'     ;
0000                P21_a           equ     b'00000000'     ;
0000                P22_x           equ     b'00000000'     ;
0000                P22_a           equ     b'00000000'     ;
0000                P23_x           equ     b'00000000'     ;
0000                P23_a           equ     b'00000000'     ;

                    ; define OR plane programming variables:
                    x
00ED                OR_a0           equ     b'11101101'     ; for output Y0
00D7                OR_b0           equ     b'11010111'     ;
0000                OR_c0           equ     b'00000000'     ;
009F                OR_a1           equ     b'10011111'     ; for output Y1
0027                OR_b1           equ     b'00100111'     ;
0000                OR_c1           equ     b'00000000'     ;
00FB                OR_a2           equ     b'11111011'     ; for output Y2
002F                OR_b2           equ     b'00101111'     ;
0000                OR_c2           equ     b'00000000'     ;
006D                OR_a3           equ     b'01101101'     ; for output Y3
0079                OR_b3           equ     b'01111001'     ;
0000                OR_c3           equ     b'00000000'     ;
0045                OR_a4           equ     b'01000101'     ; for output Y4
00FD                OR_b4           equ     b'11111101'     ;
0000                OR_c4           equ     b'00000000'     ;
0071                OR_a5           equ     b'01110001'     ; for output Y5
00DF                OR_b5           equ     b'11011111'     ;
0000                OR_c5           equ     b'00000000'     ;
007C                OR_a6           equ     b'01111100'     ; for output Y6
00EF                OR_b6           equ     b'11101111'     ;
0000                OR_c6           equ     b'00000000'     ;
0000                OR_a7           equ     b'00000000'     ; for output Y7
0000                OR_b7           equ     b'00000000'     ;
0000                OR_c7           equ     b'00000000'     ;


                                    org     01ffh           ;
01FF 0A00           begin           goto    main            ;

                                    org     000h            ;

                    ; define macro to evaluate 1 product (AND) term:
                    ;
0000 0902           main            call    pla88           ;
0001 0A00                           goto    main            ;
                    ;
```


© 1994 Microchip Technology Inc.

```
                EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                        movf    input,W         ;
                        xorlw   Pn_x            ;
                        andlw   Pn_a            ;
                        btfsc   status,bit2     ; skip if zero bit not set
                        bsf     Preg_x,bit_n    ; product term = 1
                        ENDM

                ; define macro to load OR term constants:
                ;
                EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                        LOCAL   SETBIT          ;
                        movf    Preg_a,W        ;
                        andlw   OR_an           ;
                        btfss   status,bit2     ;
                        goto    SETBIT          ;

                        movf    Preg_b,W        ;
                        andlw   OR_bn           ;
                        btfss   status,bit2     ;
                        goto    SETBIT          ;

                        movf    Preg_c,W        ;
                        andlw   OR_cn           ;
                        btfss   status,bit2     ;
                SETBIT  bsf     Y_reg,bit_n     ;
                        ENDM

                ; now the  PLA evaluation procedure:
                ;
0002 0CFF       pla88   movlw   0ffh            ;
0003 0006               tris    6               ; port_b = input
0004 0206               movf    port_b,W        ; read input
0005 002C               movwf   input           ; store input in a register
0006 006E               clrf    Preg_a          ; clear Product register a
0007 006F               clrf    Preg_b          ; clear Product register b
0008 0070               clrf    Preg_c          ; clear Product register c
0009 006D               clrf    Y_reg           ; clear output register

                        EVAL_P  Preg_a,bit0,P0_x,P0_a
000A 020C                       movf    input,W         ;
000B 0F00                       xorlw   P0_x            ;
000C 0E0F                       andlw   P0_a            ;
000D 0643                       btfsc   status,bit2     ; skip if zero bit not set
000E 050E                       bsf     Preg_a,bit0     ; product term = 1

                        EVAL_P  Preg_a,bit1,P1_x,P1_a
000F 020C                       movf    input,W         ;
0010 0F01                       xorlw   P1_x            ;
0011 0E0F                       andlw   P1_a            ;
0012 0643                       btfsc   status,bit2     ; skip if zero bit not set
0013 052E                       bsf     Preg_a,bit1     ; product term = 1

                        EVAL_P  Preg_a,bit2,P2_x,P2_a
0014 020C                       movf    input,W         ;
0015 0F02                       xorlw   P2_x            ;
0016 0E0F                       andlw   P2_a            ;
0017 0643                       btfsc   status,bit2     ; skip if zero bit not set
0018 054E                       bsf     Preg_a,bit2     ; product term = 1

                        EVAL_P  Preg_a,bit3,P3_x,P3_a
0019 020C                       movf    input,W         ;
001A 0F03                       xorlw   P3_x            ;
001B 0E0F                       andlw   P3_a            ;
001C 0643                       btfsc   status,bit2     ; skip if zero bit not set
001D 056E                       bsf     Preg_a,bit3     ; product term = 1
```

© 1994 Microchip Technology Inc.

```
                        EVAL_P  Preg_a,bit4,P4_x,P4_a
001E 020C                       movf    input,W         ;
001F 0F04                       xorlw   P4_x            ;
0020 0E0F                       andlw   P4_a            ;
0021 0643                       btfsc   status,bit2     ; skip if zero bit not set
0022 058E                       bsf     Preg_a,bit4     ; product term = 1

                        EVAL_P  Preg_a,bit5,P5_x,P5_a
0023 020C                       movf    input,W         ;
0024 0F05                       xorlw   P5_x            ;
0025 0E0F                       andlw   P5_a            ;
0026 0643                       btfsc   status,bit2     ; skip if zero bit not set
0027 05AE                       bsf     Preg_a,bit5     ; product term = 1

                        EVAL_P  Preg_a,bit6,P6_x,P6_a
0028 020C                       movf    input,W         ;
0029 0F06                       xorlw   P6_x            ;
002A 0E0F                       andlw   P6_a            ;
002B 0643                       btfsc   status,bit2     ; skip if zero bit not set
002C 05CE                       bsf     Preg_a,bit6     ; product term = 1

                        EVAL_P  Preg_a,bit7,P7_x,P7_a
002D 020C                       movf    input,W         ;
002E 0F07                       xorlw   P7_x            ;
002F 0E0F                       andlw   P7_a            ;
0030 0643                       btfsc   status,bit2     ; skip if zero bit not set
0031 05EE                       bsf     Preg_a,bit7     ; product term = 1


                        EVAL_P  Preg_b,bit0,P8_x,P8_a
0032 020C                       movf    input,W         ;
0033 0F08                       xorlw   P8_x            ;
0034 0E0F                       andlw   P8_a            ;
0035 0643                       btfsc   status,bit2     ; skip if zero bit not set
0036 050F                       bsf     Preg_b,bit0     ; product term = 1

                        EVAL_P  Preg_b,bit1,P9_x,P9_a
0037 020C                       movf    input,W         ;
0038 0F09                       xorlw   P9_x            ;
0039 0E0F                       andlw   P9_a            ;
003A 0643                       btfsc   status,bit2     ; skip if zero bit not set
003B 052F                       bsf     Preg_b,bit1     ; product term = 1

                        EVAL_P  Preg_b,bit2,P10_x,P10_a
003C 020C                       movf    input,W         ;
003D 0F0A                       xorlw   P10_x           ;
003E 0E0F                       andlw   P10_a           ;
003F 0643                       btfsc   status,bit2     ; skip if zero bit not set
0040 054F                       bsf     Preg_b,bit2     ; product term = 1

                        EVAL_P  Preg_b,bit3,P11_x,P11_a
0041 020C                       movf    input,W         ;
0042 0F0B                       xorlw   P11_x           ;
0043 0E0F                       andlw   P11_a           ;
0044 0643                       btfsc   status,bit2     ; skip if zero bit not set
0045 056F                       bsf     Preg_b,bit3     ; product term = 1

                        EVAL_P  Preg_b,bit4,P12_x,P12_a
0046 020C                       movf    input,W         ;
0047 0F0C                       xorlw   P12_x           ;
0048 0E0F                       andlw   P12_a           ;
0049 0643                       btfsc   status,bit2     ; skip if zero bit not set
004A 058F                       bsf     Preg_b,bit4     ; product term = 1

                        EVAL_P  Preg_b,bit5,P13_x,P13_a
004B 020C                       movf    input,W         ;
004C 0F0D                       xorlw   P13_x           ;
004D 0E0F                       andlw   P13_a           ;
004E 0643                       btfsc   status,bit2     ; skip if zero bit not set
004F 05AF                       bsf     Preg_b,bit5     ; product term = 1
```


© 1994 Microchip Technology Inc.

```
                        EVAL_P  Preg_b,bit6,P14_x,P14_a
0050 020C                       movf    input,W         ;
0051 0F0E                       xorlw   P14_x           ;
0052 0E0F                       andlw   P14_a           ;
0053 0643                       btfsc   status,bit2     ; skip if zero bit not set
0054 05CF                       bsf     Preg_b,bit6     ; product term = 1

                        EVAL_P  Preg_b,bit7,P15_x,P15_a
0055 020C                       movf    input,W         ;
0056 0F0F                       xorlw   P15_x           ;
0057 0E0F                       andlw   P15_a           ;
0058 0643                       btfsc   status,bit2     ; skip if zero bit not set
0059 05EF                       bsf     Preg_b,bit7     ; product term = 1


                        EVAL_P  Preg_c,bit0,P16_x,P16_a
005A 020C                       movf    input,W         ;
005B 0F00                       xorlw   P16_x           ;
005C 0E00                       andlw   P16_a           ;
005D 0643                       btfsc   status,bit2     ; skip if zero bit not set
005E 0510                       bsf     Preg_c,bit0     ; product term = 1

                        EVAL_P  Preg_c,bit1,P17_x,P17_a
005F 020C                       movf    input,W         ;
0060 0F00                       xorlw   P17_x           ;
0061 0E00                       andlw   P17_a           ;
0062 0643                       btfsc   status,bit2     ; skip if zero bit not set
0063 0530                       bsf     Preg_c,bit1     ; product term = 1

                        EVAL_P  Preg_c,bit2,P18_x,P18_a
0064 020C                       movf    input,W         ;
0065 0F00                       xorlw   P18_x           ;
0066 0E00                       andlw   P18_a           ;
0067 0643                       btfsc   status,bit2     ; skip if zero bit not set
0068 0550                       bsf     Preg_c,bit2     ; product term = 1

                        EVAL_P  Preg_c,bit3,P19_x,P19_a
0069 020C                       movf    input,W         ;
006A 0F00                       xorlw   P19_x           ;
006B 0E00                       andlw   P19_a           ;
006C 0643                       btfsc   status,bit2     ; skip if zero bit not set
006D 0570                       bsf     Preg_c,bit3     ; product term = 1

                        EVAL_P  Preg_c,bit4,P20_x,P20_a
006E 020C                       movf    input,W         ;
006F 0F00                       xorlw   P20_x           ;
0070 0E00                       andlw   P20_a           ;
0071 0643                       btfsc   status,bit2     ; skip if zero bit not set
0072 0590                       bsf     Preg_c,bit4     ; product term = 1

                        EVAL_P  Preg_c,bit5,P21_x,P21_a
0073 020C                       movf    input,W         ;
0074 0F00                       xorlw   P21_x           ;
0075 0E00                       andlw   P21_a           ;
0076 0643                       btfsc   status,bit2     ; skip if zero bit not set
0077 05B0                       bsf     Preg_c,bit5     ; product term = 1

                        EVAL_P  Preg_c,bit6,P22_x,P22_a
0078 020C                       movf    input,W         ;
0079 0F00                       xorlw   P22_x           ;
007A 0E00                       andlw   P22_a           ;
007B 0643                       btfsc   status,bit2     ; skip if zero bit not set
007C 05D0                       bsf     Preg_c,bit6     ; product term = 1

                        EVAL_P  Preg_c,bit7,P23_x,P23_a
007D 020C                       movf    input,W         ;
007E 0F00                       xorlw   P23_x           ;
007F 0E00                       andlw   P23_a           ;
0080 0643                       btfsc   status,bit2     ; skip if zero bit not set
```


© 1994 Microchip Technology Inc.

```
0081 05F0                       bsf     Preg_c,bit7     ; product term = 1
                    or_pl   EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
                            LOCAL   SETBIT          ;
0082 020E                       movf    Preg_a,W        ;
0083 0EED                       andlw   OR_a0           ;
0084 0743                       btfss   status,bit2     ;
0085 0A8D                       goto    SETBIT          ;

0086 020F                       movf    Preg_b,W        ;
0087 0ED7                       andlw   OR_b0           ;
0088 0743                       btfss   status,bit2     ;
0089 0A8D                       goto    SETBIT          ;

008A 0210                       movf    Preg_c,W        ;
008B 0E00                       andlw   OR_c0           ;
008C 0743                       btfss   status,bit2     ;
008D 050D           SETBIT      bsf     Y_reg,bit0      ;

                    EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
                            LOCAL   SETBIT          ;
008E 020E                       movf    Preg_a,W        ;
008F 0E9F                       andlw   OR_a1           ;
0090 0743                       btfss   status,bit2     ;
0091 0A99                       goto    SETBIT          ;

0092 020F                       movf    Preg_b,W        ;
0093 0E27                       andlw   OR_b1           ;
0094 0743                       btfss   status,bit2     ;
0095 0A99                       goto    SETBIT          ;

0096 0210                       movf    Preg_c,W        ;
0097 0E00                       andlw   OR_c1           ;
0098 0743                       btfss   status,bit2     ;
0099 052D           SETBIT      bsf     Y_reg,bit1      ;

                    EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
                            LOCAL   SETBIT          ;
009A 020E                       movf    Preg_a,W        ;
009B 0EFB                       andlw   OR_a2           ;
009C 0743                       btfss   status,bit2     ;
009D 0AA5                       goto    SETBIT          ;

009E 020F                       movf    Preg_b,W        ;
009F 0E2F                       andlw   OR_b2           ;
00A0 0743                       btfss   status,bit2     ;
00A1 0AA5                       goto    SETBIT          ;

00A2 0210                       movf    Preg_c,W        ;
00A3 0E00                       andlw   OR_c2           ;
00A4 0743                       btfss   status,bit2     ;
00A5 054D           SETBIT      bsf     Y_reg,bit2      ;

                    EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
                            LOCAL   SETBIT          ;
00A6 020E                       movf    Preg_a,W        ;
00A7 0E6D                       andlw   OR_a3           ;
00A8 0743                       btfss   status,bit2     ;
00A9 0AB1                       goto    SETBIT          ;

00AA 020F                       movf    Preg_b,W        ;
00AB 0E79                       andlw   OR_b3           ;
00AC 0743                       btfss   status,bit2     ;
00AD 0AB1                       goto    SETBIT          ;

00AE 0210                       movf    Preg_c,W        ;
00AF 0E00                       andlw   OR_c3           ;
00B0 0743                       btfss   status,bit2     ;
00B1 056D           SETBIT      bsf     Y_reg,bit3      ;
```


© 1994 Microchip Technology Inc.

```
                        EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
                            LOCAL   SETBIT              ;
00B2 020E                     movf    Preg_a,W          ;
00B3 0E45                     andlw   OR_a4             ;
00B4 0743                     btfss   status,bit2       ;
00B5 0ABD                     goto    SETBIT            ;

00B6 020F                     movf    Preg_b,W          ;
00B7 0EFD                     andlw   OR_b4             ;
00B8 0743                     btfss   status,bit2       ;
00B9 0ABD                     goto    SETBIT            ;

00BA 0210                     movf    Preg_c,W          ;
00BB 0E00                     andlw   OR_c4             ;
00BC 0743                     btfss   status,bit2       ;
00BD 058D           SETBIT    bsf     Y_reg,bit4        ;

                        EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
                            LOCAL   SETBIT              ;
00BE 020E                     movf    Preg_a,W          ;
00BF 0E71                     andlw   OR_a5             ;
00C0 0743                     btfss   status,bit2       ;
00C1 0AC9                     goto    SETBIT            ;

00C2 020F                     movf    Preg_b,W          ;
00C3 0EDF                     andlw   OR_b5             ;
00C4 0743                     btfss   status,bit2       ;
00C5 0AC9                     goto    SETBIT            ;

00C6 0210                     movf    Preg_c,W          ;
00C7 0E00                     andlw   OR_c5             ;
00C8 0743                     btfss   status,bit2       ;
00C9 05AD           SETBIT    bsf     Y_reg,bit5        ;

                        EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
                            LOCAL   SETBIT              ;
00CA 020E                     movf    Preg_a,W          ;
00CB 0E7C                     andlw   OR_a6             ;
00CC 0743                     btfss   status,bit2       ;
00CD 0AD5                     goto    SETBIT            ;

00CE 020F                     movf    Preg_b,W          ;
00CF 0EEF                     andlw   OR_b6             ;
00D0 0743                     btfss   status,bit2       ;
00D1 0AD5                     goto    SETBIT            ;

00D2 0210                     movf    Preg_c,W          ;
00D3 0E00                     andlw   OR_c6             ;
00D4 0743                     btfss   status,bit2       ;
00D5 05CD           SETBIT    bsf     Y_reg,bit6        ;

                        EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
                            LOCAL   SETBIT              ;
00D6 020E                     movf    Preg_a,W          ;
00D7 0E00                     andlw   OR_a7             ;
00D8 0743                     btfss   status,bit2       ;
00D9 0AE1                     goto    SETBIT            ;

00DA 020F                     movf    Preg_b,W          ;
00DB 0E00                     andlw   OR_b7             ;
00DC 0743                     btfss   status,bit2       ;
00DD 0AE1                     goto    SETBIT            ;

00DE 0210                     movf    Preg_c,W          ;
00DF 0E00                     andlw   OR_c7             ;
00E0 0743                     btfss   status,bit2       ;
00E1 05ED           SETBIT    bsf     Y_reg,bit7        ;
```


© 1994 Microchip Technology Inc.

```
                        ; Y_reg now contains 8 output values:
00E2 0040               wr_out  clrw                    ;
00E3 0007                       tris    7               ; port_c = output
00E4 020D                       movf    Y_reg,W         ;
00E5 0027                       movwf   port_c          ; Y_reg -> port_c
00E6 0800                       retlw   0               ;

00E7 0000               ZZZ             nop

                                END


Errors   :     0
Warnings :     0
```

APPENDIX B: PLA IMPLEMENTATION: CODE EFFICIENT APPROACH

```
MPASM B0.54                                                     PAGE  1



                        ;*********************************************************************
                        ; pla1b.asm :
                        ; This procedure implements a simple AND-OR PLA with:
                        ;
                        ;       8  inputs         := A7 A6 A5 A4 A3 A2 A1 A0
                        ;       24 product terms  := P23 P22 ..... P0
                        ;       8  outputs        := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                        ;
                        ; The eight inputs are assumed to be connected to PORT RB such that
                        ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                        ; The outputs are programmed to appear on port RC such that
                        ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                        ;
                        ; This implementation optimizes program memory usage over
                        ; speed
                        ;*********************************************************************
                        ;
                        ; define RAM locations used:
                        ;
                                LIST    P=16C57
000C                    input           equ     d"12"   ; RAM location 12 holds input
000D                    Y_reg           equ     d"13"   ; holds output result

000E                    Preg_a          equ     d"14"   ; Product terms P0 to P7. Preg_a<0> = P0
000F                    Preg_b          equ     d"15"   ; Product terms P8 to P15. Preg_b<0> = P8
0010                    Preg_c          equ     d"16"   ; Product terms P16 to P23. Preg_c<0> = P16
0012                    Pn_x            equ     d"18"   ;
0013                    Pn_a            equ     d"19"   ;
0014                    OR_a            equ     d"20"   ;
0015                    OR_b            equ     d"21"   ;
0016                    OR_c            equ     d"22"   ;

                        ; define some constants and file addresses:
                        ;
0000                    bit0            equ     0       ;
0001                    bit1            equ     1       ;
0002                    bit2            equ     2       ;
0003                    bit3            equ     3       ;
0004                    bit4            equ     4       ;
0005                    bit5            equ     5       ;
0006                    bit6            equ     6       ;
0007                    bit7            equ     7       ;
                        ;
0003                    status          equ     3       ;
0006                    port_b          equ     6       ;
0007                    port_c          equ     7       ;
```


© 1994 Microchip Technology Inc.

```
                        ;
                        ; define the AND plane programming variables:
                        ;
0000                    P0_x            equ     b"00000000"     ;
000F                    P0_a            equ     b"00001111"     ;
0001                    P1_x            equ     b"00000001"     ;
000F                    P1_a            equ     b"00001111"     ;
0002                    P2_x            equ     b"00000010"     ;
000F                    P2_a            equ     b"00001111"     ;
0003                    P3_x            equ     b"00000011"     ;
000F                    P3_a            equ     b"00001111"     ;
0004                    P4_x            equ     b"00000100"     ;
000F                    P4_a            equ     b"00001111"     ;
0005                    P5_x            equ     b"00000101"     ;
000F                    P5_a            equ     b"00001111"     ;
0006                    P6_x            equ     b"00000110"     ;
000F                    P6_a            equ     b"00001111"     ;
0007                    P7_x            equ     b"00000111"     ;
000F                    P7_a            equ     b"00001111"     ;
0008                    P8_x            equ     b"00001000"     ;
000F                    P8_a            equ     b"00001111"     ;
0009                    P9_x            equ     b"00001001"     ;
000F                    P9_a            equ     b"00001111"     ;
000A                    P10_x           equ     b"00001010"     ;
000F                    P10_a           equ     b"00001111"     ;
000B                    P11_x           equ     b"00001011"     ;
000F                    P11_a           equ     b"00001111"     ;
000C                    P12_x           equ     b"00001100"     ;
000F                    P12_a           equ     b"00001111"     ;
000D                    P13_x           equ     b"00001101"     ;
000F                    P13_a           equ     b"00001111"     ;
000E                    P14_x           equ     b"00001110"     ;
000F                    P14_a           equ     b"00001111"     ;
000F                    P15_x           equ     b"00001111"     ;
000F                    P15_a           equ     b"00001111"     ;
0000                    P16_x           equ     b"00000000"     ;
0000                    P16_a           equ     b"00000000"     ;
0000                    P17_x           equ     b"00000000"     ;
0000                    P17_a           equ     b"00000000"     ;
0000                    P18_x           equ     b"00000000"     ;
0000                    P18_a           equ     b"00000000"     ;
0000                    P19_x           equ     b"00000000"     ;
0000                    P19_a           equ     b"00000000"     ;
0000                    P20_x           equ     b"00000000"     ;
0000                    P20_a           equ     b"00000000"     ;
0000                    P21_x           equ     b"00000000"     ;
0000                    P21_a           equ     b"00000000"     ;
0000                    P22_x           equ     b"00000000"     ;
0000                    P22_a           equ     b"00000000"     ;
0000                    P23_x           equ     b"00000000"     ;
0000                    P23_a           equ     b"00000000"     ;

                        ; define OR plane programming variables:

00ED                    OR_a0           equ     b"11101101"     ; for output Y0
00D7                    OR_b0           equ     b"11010111"     ;
0000                    OR_c0           equ     b"00000000"     ;
009F                    OR_a1           equ     b"10011111"     ; for output Y1
0027                    OR_b1           equ     b"00100111"     ;
0000                    OR_c1           equ     b"00000000"     ;
00FB                    OR_a2           equ     b"11111011"     ; for output Y2
002F                    OR_b2           equ     b"00101111"     ;
0000                    OR_c2           equ     b"00000000"     ;
006D                    OR_a3           equ     b"01101101"     ; for output Y3
0079                    OR_b3           equ     b"01111001"     ;
0000                    OR_c3           equ     b"00000000"     ;
0045                    OR_a4           equ     b"01000101"     ; for output Y4
00FD                    OR_b4           equ     b"11111101"     ;
0000                    OR_c4           equ     b"00000000"     ;
```

© 1994 Microchip Technology Inc.

```
0071                OR_a5           equ     b"01110001"     ; for output Y5
00DF                OR_b5           equ     b"11011111"     ;
0000                OR_c5           equ     b"00000000"     ;
007C                OR_a6           equ     b"01111100"     ; for output Y6
00EF                OR_b6           equ     b"11101111"     ;
0000                OR_c6           equ     b"00000000"     ;
0000                OR_a7           equ     b"00000000"     ; for output Y7
0000                OR_b7           equ     b"00000000"     ;
0000                OR_c7           equ     b"00000000"     ;


                                    org     01ffh           ;
01FF 0A00           begin           goto    main            ;

                                    org     000h            ;
                    ; define macro to evaluate 1 product (AND) term:
0000 090B           main            call    pla88           ;
0001 0A00                           goto    main            ;
                    ;
                    EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                              movf    input,W               ;
                              xorlw   Pn_x                  ;
                              andlw   Pn_a                  ;
                              btfsc   status,bit2           ; skip if zero bit not set
                              bsf     Preg_x,bit_n          ; product term = 1
                              ENDM


                    ; define macro to load OR term constants:
                    ;
                    EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                              movlw   OR_an                 ; load constants
                              movwf   OR_a                  ;
                              movlw   OR_bn                 ;
                              movwf   OR_b                  ;
                              movlw   OR_cn                 ;
                              movwf   OR_c                  ;
                              call    EVAL1                 ;
                              btfss   status,bit2           ;
                              bsf     Y_reg,bit_n           ;
                              ENDM
                    ; define procedure to evaluate 1 output (OR) term:
                    ;
0002 020E           EVAL1             movf    Preg_a,W      ;
0003 0174                             andwf   OR_a,1        ;

0004 020F                             movf    Preg_b,W      ;
0005 0175                             andwf   OR_b,1        ;

0006 0210                             movf    Preg_c,W      ;
0007 0156                             andwf   OR_c,W        ;

0008 0114                             iorwf   OR_a,W        ;
0009 0115                             iorwf   OR_b,W        ;
000A 0800                             retlw   0             ; W = 1 implies Yn = 1

                    ; now the  PLA evaluation procedure:
                    ;
000B 0CFF           pla88             movlw   0ffh          ;
000C 0006                             tris    6             ; port_b = input
000D 0206                             movf    port_b,W      ; read input
000E 002C                             movwf   input         ; store input in a register
000F 006E                             clrf    Preg_a        ; clear Product register a
0010 006F                             clrf    Preg_b        ; clear Product register b
0011 0070                             clrf    Preg_c        ; clear Product register c
0012 006D                             clrf    Y_reg         ; clear output register
```


© 1994 Microchip Technology Inc.

```
                and_pl      EVAL_P  Preg_a,bit0,P0_x,P0_a
0013 020C                           movf    input,W      ;
0014 0F00                           xorlw   P0_x         ;
0015 0E0F                           andlw   P0_a         ;
0016 0643                           btfsc   status,bit2  ; skip if zero bit not set
0017 050E                           bsf     Preg_a,bit0  ; product term = 1

                                    EVAL_P  Preg_a,bit1,P1_x,P1_a
0018 020C                           movf    input,W      ;
0019 0F01                           xorlw   P1_x         ;
001A 0E0F                           andlw   P1_a         ;
001B 0643                           btfsc   status,bit2  ; skip if zero bit not set
001C 052E                           bsf     Preg_a,bit1  ; product term = 1

                                    EVAL_P  Preg_a,bit2,P2_x,P2_a
001D 020C                           movf    input,W      ;
001E 0F02                           xorlw   P2_x         ;
001F 0E0F                           andlw   P2_a         ;
0020 0643                           btfsc   status,bit2  ; skip if zero bit not set
0021 054E                           bsf     Preg_a,bit2  ; product term = 1

                                    EVAL_P  Preg_a,bit3,P3_x,P3_a
0022 020C                           movf    input,W      ;
0023 0F03                           xorlw   P3_x         ;
0024 0E0F                           andlw   P3_a         ;
0025 0643                           btfsc   status,bit2  ; skip if zero bit not set
0026 056E                           bsf     Preg_a,bit3  ; product term = 1

                                    EVAL_P  Preg_a,bit4,P4_x,P4_a
0027 020C                           movf    input,W      ;
0028 0F04                           xorlw   P4_x         ;
0029 0E0F                           andlw   P4_a         ;
002A 0643                           btfsc   status,bit2  ; skip if zero bit not set
002B 058E                           bsf     Preg_a,bit4  ; product term = 1

                                    EVAL_P  Preg_a,bit5,P5_x,P5_a
002C 020C                           movf    input,W      ;
002D 0F05                           xorlw   P5_x         ;
002E 0E0F                           andlw   P5_a         ;
002F 0643                           btfsc   status,bit2  ; skip if zero bit not set
0030 05AE                           bsf     Preg_a,bit5  ; product term = 1

                                    EVAL_P  Preg_a,bit6,P6_x,P6_a
0031 020C                           movf    input,W      ;
0032 0F06                           xorlw   P6_x         ;
0033 0E0F                           andlw   P6_a         ;
0034 0643                           btfsc   status,bit2  ; skip if zero bit not set
0035 05CE                           bsf     Preg_a,bit6  ; product term = 1

                                    EVAL_P  Preg_a,bit7,P7_x,P7_a
0036 020C                           movf    input,W      ;
0037 0F07                           xorlw   P7_x         ;
0038 0E0F                           andlw   P7_a         ;
0039 0643                           btfsc   status,bit2  ; skip if zero bit not set
003A 05EE                           bsf     Preg_a,bit7  ; product term = 1


                                    EVAL_P  Preg_b,bit0,P8_x,P8_a
003B 020C                           movf    input,W      ;
003C 0F08                           xorlw   P8_x         ;
003D 0E0F                           andlw   P8_a         ;
003E 0643                           btfsc   status,bit2  ; skip if zero bit not set
003F 050F                           bsf     Preg_b,bit0  ; product term = 1

                                    EVAL_P  Preg_b,bit1,P9_x,P9_a
0040 020C                           movf    input,W      ;
0041 0F09                           xorlw   P9_x         ;
0042 0E0F                           andlw   P9_a         ;
0043 0643                           btfsc   status,bit2  ; skip if zero bit not set
0044 052F                           bsf     Preg_b,bit1  ; product term = 1
```


© 1994 Microchip Technology Inc.

```
                        EVAL_P  Preg_b,bit2,P10_x,P10_a
0045 020C                       movf    input,W         ;
0046 0F0A                       xorlw   P10_x           ;
0047 0E0F                       andlw   P10_a           ;
0048 0643                       btfsc   status,bit2     ; skip if zero bit not set
0049 054F                       bsf     Preg_b,bit2     ; product term = 1

                        EVAL_P  Preg_b,bit3,P11_x,P11_a
004A 020C                       movf    input,W         ;
004B 0F0B                       xorlw   P11_x           ;
004C 0E0F                       andlw   P11_a           ;
004D 0643                       btfsc   status,bit2     ; skip if zero bit not set
004E 056F                       bsf     Preg_b,bit3     ; product term = 1

                        EVAL_P  Preg_b,bit4,P12_x,P12_a
004F 020C                       movf    input,W         ;
0050 0F0C                       xorlw   P12_x           ;
0051 0E0F                       andlw   P12_a           ;
0052 0643                       btfsc   status,bit2     ; skip if zero bit not set
0053 058F                       bsf     Preg_b,bit4     ; product term = 1
                        EVAL_P  Preg_b,bit5,P13_x,P13_a
0054 020C                       movf    input,W         ;
0055 0F0D                       xorlw   P13_x           ;
0056 0E0F                       andlw   P13_a           ;
0057 0643                       btfsc   status,bit2     ; skip if zero bit not set
0058 05AF                       bsf     Preg_b,bit5     ; product term = 1

                        EVAL_P  Preg_b,bit6,P14_x,P14_a
0059 020C                       movf    input,W         ;
005A 0F0E                       xorlw   P14_x           ;
005B 0E0F                       andlw   P14_a           ;
005C 0643                       btfsc   status,bit2     ; skip if zero bit not set
005D 05CF                       bsf     Preg_b,bit6     ; product term = 1

                        EVAL_P  Preg_b,bit7,P15_x,P15_a
005E 020C                       movf    input,W         ;
005F 0F0F                       xorlw   P15_x           ;
0060 0E0F                       andlw   P15_a           ;
0061 0643                       btfsc   status,bit2     ; skip if zero bit not set
0062 05EF                       bsf     Preg_b,bit7     ; product term = 1


                        EVAL_P  Preg_c,bit0,P16_x,P16_a
0063 020C                       movf    input,W         ;
0064 0F00                       xorlw   P16_x           ;
0065 0E00                       andlw   P16_a           ;
0066 0643                       btfsc   status,bit2     ; skip if zero bit not set
0067 0510                       bsf     Preg_c,bit0     ; product term = 1

                        EVAL_P  Preg_c,bit1,P17_x,P17_a
0068 020C                       movf    input,W         ;
0069 0F00                       xorlw   P17_x           ;
006A 0E00                       andlw   P17_a           ;
006B 0643                       btfsc   status,bit2     ; skip if zero bit not set
006C 0530                       bsf     Preg_c,bit1     ; product term = 1

                        EVAL_P  Preg_c,bit2,P18_x,P18_a
006D 020C                       movf    input,W         ;
006E 0F00                       xorlw   P18_x           ;
006F 0E00                       andlw   P18_a           ;
0070 0643                       btfsc   status,bit2     ; skip if zero bit not set
0071 0550                       bsf     Preg_c,bit2     ; product term = 1

                        EVAL_P  Preg_c,bit3,P19_x,P19_a
0072 020C                       movf    input,W         ;
0073 0F00                       xorlw   P19_x           ;
0074 0E00                       andlw   P19_a           ;
0075 0643                       btfsc   status,bit2     ; skip if zero bit not set
0076 0570                       bsf     Preg_c,bit3     ; product term = 1
```


© 1994 Microchip Technology Inc.

```
                        EVAL_P  Preg_c,bit4,P20_x,P20_a
0077 020C                       movf    input,W     ;
0078 0F00                       xorlw   P20_x       ;
0079 0E00                       andlw   P20_a       ;
007A 0643                       btfsc   status,bit2 ; skip if zero bit not set
007B 0590                       bsf     Preg_c,bit4 ; product term = 1

                        EVAL_P  Preg_c,bit5,P21_x,P21_a
007C 020C                       movf    input,W     ;
007D 0F00                       xorlw   P21_x       ;
007E 0E00                       andlw   P21_a       ;
007F 0643                       btfsc   status,bit2 ; skip if zero bit not set
0080 05B0                       bsf     Preg_c,bit5 ; product term = 1

                        EVAL_P  Preg_c,bit6,P22_x,P22_a
0081 020C                       movf    input,W     ;
0082 0F00                       xorlw   P22_x       ;
0083 0E00                       andlw   P22_a       ;
0084 0643                       btfsc   status,bit2 ; skip if zero bit not set
0085 05D0                       bsf     Preg_c,bit6 ; product term = 1

                        EVAL_P  Preg_c,bit7,P23_x,P23_a
0086 020C                       movf    input,W     ;
0087 0F00                       xorlw   P23_x       ;
0088 0E00                       andlw   P23_a       ;
0089 0643                       btfsc   status,bit2 ; skip if zero bit not set
008A 05F0                       bsf     Preg_c,bit7 ; product term = 1


                or_pl    EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
008B 0CED                       movlw   OR_a0       ; load constants
008C 0034                       movwf   OR_a        ;
008D 0CD7                       movlw   OR_b0       ;
008E 0035                       movwf   OR_b        ;
008F 0C00                       movlw   OR_c0       ;
0090 0036                       movwf   OR_c        ;
0091 0902                       call    EVAL1       ;
0092 0743                       btfss   status,bit2 ;
0093 050D                       bsf     Y_reg,bit0  ;

                        EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
0094 0C9F                       movlw   OR_a1       ; load constants
0095 0034                       movwf   OR_a        ;
0096 0C27                       movlw   OR_b1       ;
0097 0035                       movwf   OR_b        ;
0098 0C00                       movlw   OR_c1       ;
0099 0036                       movwf   OR_c        ;
009A 0902                       call    EVAL1       ;
009B 0743                       btfss   status,bit2 ;
009C 052D                       bsf     Y_reg,bit1  ;

                        EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
009D 0CFB                       movlw   OR_a2       ; load constants
009E 0034                       movwf   OR_a        ;
009F 0C2F                       movlw   OR_b2       ;
00A0 0035                       movwf   OR_b        ;
00A1 0C00                       movlw   OR_c2       ;
00A2 0036                       movwf   OR_c        ;
00A3 0902                       call    EVAL1       ;
00A4 0743                       btfss   status,bit2 ;
```

© 1994 Microchip Technology Inc.

```
00A5 054D                       bsf     Y_reg,bit2      ;

                        EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
00A6 0C6D                       movlw   OR_a3           ; load constants
00A7 0034                       movwf   OR_a            ;
00A8 0C79                       movlw   OR_b3           ;
00A9 0035                       movwf   OR_b            ;
00AA 0C00                       movlw   OR_c3           ;
00AB 0036                       movwf   OR_c            ;
00AC 0902                       call    EVAL1           ;
00AD 0743                       btfss   status,bit2     ;
00AE 056D                       bsf     Y_reg,bit3      ;

                        EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
00AF 0C45                       movlw   OR_a4           ; load constants
00B0 0034                       movwf   OR_a            ;
00B1 0CFD                       movlw   OR_b4           ;
00B2 0035                       movwf   OR_b            ;
00B3 0C00                       movlw   OR_c4           ;
00B4 0036                       movwf   OR_c            ;
00B5 0902                       call    EVAL1           ;
00B6 0743                       btfss   status,bit2     ;
00B7 058D                       bsf     Y_reg,bit4      ;

                        EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
00B8 0C71                       movlw   OR_a5           ; load constants
00B9 0034                       movwf   OR_a            ;
00BA 0CDF                       movlw   OR_b5           ;
00BB 0035                       movwf   OR_b            ;
00BC 0C00                       movlw   OR_c5           ;
00BD 0036                       movwf   OR_c            ;
00BE 0902                       call    EVAL1           ;
00BF 0743                       btfss   status,bit2     ;
00C0 05AD                       bsf     Y_reg,bit5      ;

                        EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
00C1 0C7C                       movlw   OR_a6           ; load constants
00C2 0034                       movwf   OR_a            ;
00C3 0CEF                       movlw   OR_b6           ;
00C4 0035                       movwf   OR_b            ;
00C5 0C00                       movlw   OR_c6           ;
00C6 0036                       movwf   OR_c            ;
00C7 0902                       call    EVAL1           ;
00C8 0743                       btfss   status,bit2     ;
00C9 05CD                       bsf     Y_reg,bit6      ;

                        EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
00CA 0C00                       movlw   OR_a7           ; load constants
00CB 0034                       movwf   OR_a            ;
00CC 0C00                       movlw   OR_b7           ;
00CD 0035                       movwf   OR_b            ;
00CE 0C00                       movlw   OR_c7           ;
00CF 0036                       movwf   OR_c            ;
00D0 0902                       call    EVAL1           ;
00D1 0743                       btfss   status,bit2     ;
00D2 05ED                       bsf     Y_reg,bit7      ;


                ; Y_reg now contains 8 output values:
00D3 0040       wr_out          clrw                    ;
00D4 0007                       tris    7               ; port_c = output
00D5 020D                       movf    Y_reg,W         ;
00D6 0027                       movwf   port_c          ; Y_reg -> port_c
00D7 0800                       retlw   0               ;

00D8 0000       ZZZ             nop

                        END
Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN593

Serial Port Routines Without Using the RTCC

INTRODUCTION

The PIC16C5X has one 8-bit timer (RTCC) which can use an 8-bit prescaler. In some instances, the user would like to use this timer for some other purposes and yet be able to do a transmit and receive using the serial port. This application note offers routines to do a simple 8-bit transmit and receive with no handshake, at baud rates from 1200 to 9600. Please note that these routines use a timed loop which is as accurate as the clock which drives the PIC16C5X. The user enters the frequency and baud rate desired. The calculated value "delay" in the serial routine has to be an 8-bit value only. If the value is greater than 8-bits, the frequency and baud rate values have to be changed.

CONCLUSION

Simple transmit and receive routines can be written without using RTCC to generate the baud rate.

Author: Stan D'Souza
Logic Products Division

© 1994 Microchip Technology Inc.

APPENDIX A

```
MPASM 00.00.66 Beta              6-24-1994  7:4:48                    PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;
                     0002 ;These routines were written to work on the PICDEM1 hardware.
                     0003 ; The frequency of the clock is 16 Mhz and the hardware uses no
                     0004 ;handshake
                     0005 ;          TX -> RA3
                     0006 ;          RX -> RA2
                     0007            list p=16c54,f=inhx8m
                     0008 ;
00F4 2400            0009 clockrate equ .16000000
2580                 0010 baudrate equ    .9600
                     0011 ;
003D 0900            0012 fclk  equ     clockrate/4
                     0013 ;*****************************************************
                     0014 ;The value baudconst must be a 8 bit value only
0088                 0015 baudconst       equ     ((fclk/baudrate)/3 - 2)
                     0016 ;*****************************************************
0010                 0017 count   equ     0x10
0011                 0018 txreg   equ     0x11
0011                 0019 rcreg   equ     0x11
0012                 0020 delay   equ     0x12
0013                 0021 tempa   equ     0x13
0010                 0022 hi      equ     0x10
0011                 0023 lo      equ     0x11
0015                 0024 gpram   equ     0x15
                     0025 ;
                     0026            include "pic5x.h"
                     0001 ;This is the common header file for all PIC16C5X parts.
                     0002 ;
                     0003 ;
                     0004            CBLOCK 0x00
0000 0005            0005                    _indf, _rtcc, _pcl, _status, _fsr
0005 0002            0006                    _porta, _portb
                     0007            ENDC
                     0008 ;
                     0009 ; Porta Bits
0001                 0010  #define          _ra0            _porta,0
0002                 0011  #define          _ra1            _porta,1
0003                 0012  #define          _ra2            _porta,2
0004                 0013  #define          _ra3            _porta,3
                     0014
                     0015
                     0016 ; Portb bits
0005                 0017  #define          _rb0            _portb,0
0006                 0018  #define          _rb1            _portb,1
0007                 0019  #define          _rb2            _portb,2
0008                 0020  #define          _rb3            _portb,3
0009                 0021  #define          _rb4            _portb,4
000A                 0022  #define          _rb5            _portb,5
000B                 0023  #define          _rb6            _portb,6
000C                 0024  #define          _rb7            _portb,7
                     0025 ;
                     0026 ; STATUS Reg Bits
000D                 0027  #define          _carry          _status,0
000E                 0028  #define          _c              _status,0
000F                 0029  #define          _dc             _status,1
0010                 0030  #define          _z              _status,2
0011                 0031  #define          _pd             _status,3
0012                 0032  #define          _to             _status,4
0013                 0033  #define          _pa0            _status,5
0014                 0034  #define          _pa1            _status,6
                     0035 ;
                     0026
```


© 1994 Microchip Technology Inc.

```
                        0027 ;
0015                    0028  #define _tx     _porta,3
0016                    0029  #define _rx     _porta,2
                        0030 ;
                        0031          org     0
                        0032 start
0000 095A               0033          call    wait
0001 0C0F               0034          movlw   B'00001111'
0002 0025               0035          movwf   _porta
0003 0C07               0036          movlw   B'00000111'
0004 0005               0037          tris    _porta
0005 0066               0038          clrf    _portb
0006 0040               0039          clrw
0007 0006               0040          tris    _portb
0008 0CA5               0041          movlw   0xa5
0009 0195               0042          xorwf   gpram,w
000A 0643               0043          btfsc   _z
000B 0964               0044          call    Mclr
000C 0CA5               0045          movlw   0xa5
000D 0035               0046          movwf   gpram
                        0047 ;
                        0048 ;
000E 0C2F               0049          movlw   0x2f
000F 0033               0050          movwf   tempa
0010 0C38               0051          movlw   B'00111000'
0011 0002               0052          option
0012 0061               0053          clrf    _rtcc
                        0054 S1check
0013 0201               0055          movf    _rtcc,w
0014 0643               0056          btfsc   _z              ;if S1 pressed then skip
0015 0A13               0057          goto    S1check
                        0058 ;
                        0059 next
0016 02B3               0060          incf    tempa
0017 06F3               0061          btfsc   tempa,7
0018 0A61               0062          goto    AllDone
0019 0213               0063          movf    tempa,w
001A 0920               0064          call    transmit
001B 0937               0065          call    receive
001C 0093               0066          subwf   tempa,w
001D 0643               0067          btfsc   _z
001E 0A47               0068          goto    fail
001F 0A16               0069          goto    next
                        0070 ;
                        0071 ;
                        0072 transmit
0020 0031               0073          movwf   txreg
0021 0465               0074          bcf     _tx             ;send start bit
0022 0C88               0075          movlw   baudconst
0023 0032               0076          movwf   delay
0024 0C09               0077          movlw   .9
0025 0030               0078          movwf   count
                        0079 txbaudwait
0026 02F2               0080          decfsz  delay
0027 0A26               0081          goto    txbaudwait
0028 0C88               0082          movlw   baudconst
0029 0032               0083          movwf   delay
002A 02F0               0084          decfsz  count
002B 0A30               0085          goto    SendNextBit
002C 0C09               0086          movlw   .9
002D 0030               0087          movwf   count
002E 0565               0088          bsf     _tx             ;send stop bit
002F 0800               0089          return
                        0090 SendNextBit
0030 0331               0091          rrf     txreg
0031 0703               0092          btfss   _c
0032 0A35               0093          goto    Setlo
0033 0565               0094          bsf     _tx
0034 0A26               0095          goto    txbaudwait
```

© 1994 Microchip Technology Inc.

```
                        0096 Setlo
0035 0465               0097         bcf     _tx
0036 0A26               0098         goto    txbaudwait
                        0099 ;
                        0100 ;
                        0101 receive
0037 0645               0102        btfsc    _rx
0038 0A37               0103        goto     receive        ;wait for receive
                        0104 rxbaudwait
0039 02F2               0105         decfsz  delay
003A 0A39               0106         goto    rxbaudwait
003B 0C88               0107         movlw   baudconst
003C 0032               0108         movwf   delay
003D 02F0               0109         decfsz  count
003E 0A42               0110         goto    RecvNextBit
003F 0C09               0111         movlw   .9
0040 0030               0112         movwf   count
0041 0800               0113         return
                        0114 RecvNextBit
0042 0403               0115         bcf     _c
0043 0645               0116         btfsc   _rx
0044 0503               0117         bsf     _c
0045 0331               0118         rrf     rcreg
0046 0A39               0119         goto    rxbaudwait
                        0120 ;
                        0121 fail
0047 0266               0122         comf    _portb
0048 094A               0123         call    halfsec
0049 0A47               0124         goto    fail
                        0125 halfsec
004A 0070               0126         clrf    hi
004B 0071               0127         clrf    lo
                        0128 hsloop
004C 0000               0129         nop
004D 0000               0130         nop
004E 0000               0131         nop
004F 0000               0132         nop
0050 0000               0133         nop
0051 0000               0134         nop
0052 0000               0135         nop
0053 0000               0136         nop
0054 0000               0137         nop
0055 02F1               0138         decfsz  lo
0056 0A4C               0139         goto    hsloop
0057 02F0               0140         decfsz  hi
0058 0A4C               0141         goto    hsloop
0059 0800               0142         return
                        0143 ;
                        0144 wait
005A 0070               0145         clrf    hi
005B 0071               0146         clrf    lo
                        0147 dly
005C 02F1               0148         decfsz  lo
005D 0A5C               0149         goto    dly
005E 02F0               0150         decfsz  hi
005F 0A5C               0151         goto    dly
0060 0800               0152         return
                        0153
                        0154 ;
                        0155 AllDone
0061 0C55               0156         movlw   0x55
0062 0026               0157         movwf   _portb
0063 0A63               0158         goto    $
                        0159 ;
                        0160 Mclr
0064 0066               0161         clrf    _portb
0065 00E6               0162         decf    _portb
0066 0075               0163         clrf    gpram
                        0164 S3check
```


© 1994 Microchip Technology Inc.

```
0067 0625           0165        btfsc   _ra1
0068 0A67           0166        goto    S3check
0069 0066           0167        clrf    _portb
006A 0A6A           0168        goto    $
                    0169 ;
                    0170 ;
                    0171        org     0x1ff
01FF 0A00           0172        goto    start
                    0173 ;
                    0174        end
                    0175
                    0176
                    0177
                    0178
                    0179
                    0180
                    0181

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX---- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

AN510

Implementation of an Asynchronous Serial I/O

INTRODUCTION

The PIC16C5X series from Microchip Technology Inc., are 8-bit, high-speed, EPROM-based microcontrollers. This application note describes the implementation of an Asynchronous serial I/O using Microchip's PIC16C5X series of high-speed 8-bit microcontrollers. These EPROM based microcontrollers can operate at very high speeds with a minimum of 200 ns cycle time @ 20 MHz input clock. Many microcontroller applications require chip-to-chip serial data communications. Since the PIC16C5X series have no on chip serial ports, serial communication has to be performed in software. For many cost-sensitive high volume applications, implementation of a serial I/O through software provides a more cost effective solution than dedicated logic. This application note provides code for PIC16C5X to simulate a serial port using two I/O Pins (one as input for reception and the other as output for transmission).

IMPLEMENTATION

Two programs are provided in this application note. One program simulates a full duplex RS-232 communication and the other provides implementation of half duplex communication. Using Half-Duplex, rates up to 19200 baud can be implemented using an 8 MHz input clock. In case of Full-Duplex, the software can handle up to 9600 baud at 8 MHz and 19200 baud at 20 MHz, one or two stop bits, eight or seven data bits, No Parity and can transmit or receive with either LSB first (normal mode) or MSB first (CODEC like mode). It should be noted that the higher the input clock the better the resolution. These options should be set up during assembly time and not during run time. The user simply has to change the header file for the required communication options. The software does not provide any handshaking protocols. With minor modifications, the user may incorporate software handshaking using XON/XOFF. To implement hardware handshaking, an additional two digital I/O Pins may be used as RTS (ready to send) and CTS (clear to send) lines.

Figure 1 shows a flow chart for serial transmission and Figure 2 shows a flow chart for reception. The flowcharts show cases for transmission/reception with LSB first and eight data bits. For reception, the data receive pin, DR, is polled approximately every B/2 seconds (52 µs in case of 9600 baud) to detect the start bit, where B is the time duration of one bit (B = 1/Baud). If a start bit is found, then the first data bit is checked for after 1.25B seconds. From then on, the other data bits are checked every B seconds (104µs in case of 9600 baud).

In the case of transmission, first a start bit is sent by setting the transmit data pin, DX to zero for B seconds, and from then on the DX pin is set/cleared corresponding to the data bit every B seconds. Assembly language code corresponding to the following flowcharts is given in Figures 3 and 4.

© 1994 Microchip Technology Inc.

FIGURE 1 - TRANSMISSION FLOW CHART

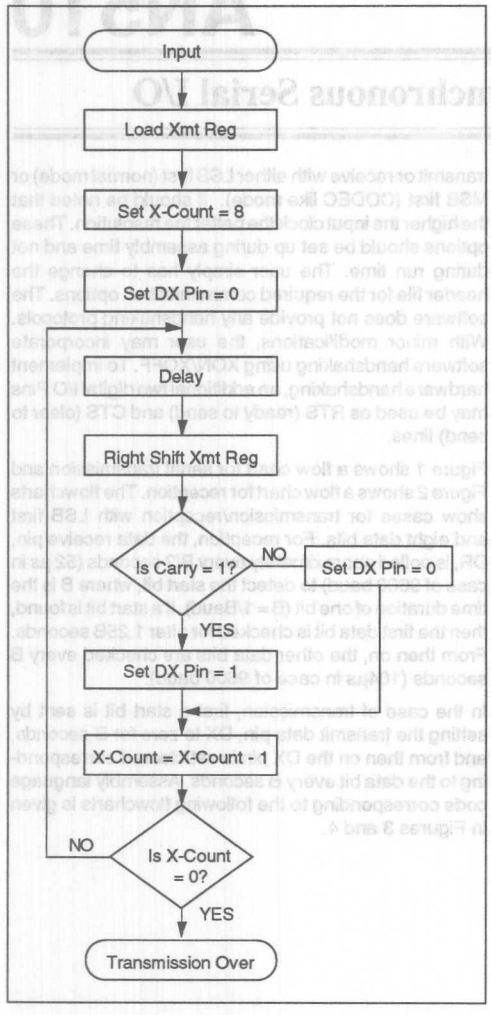
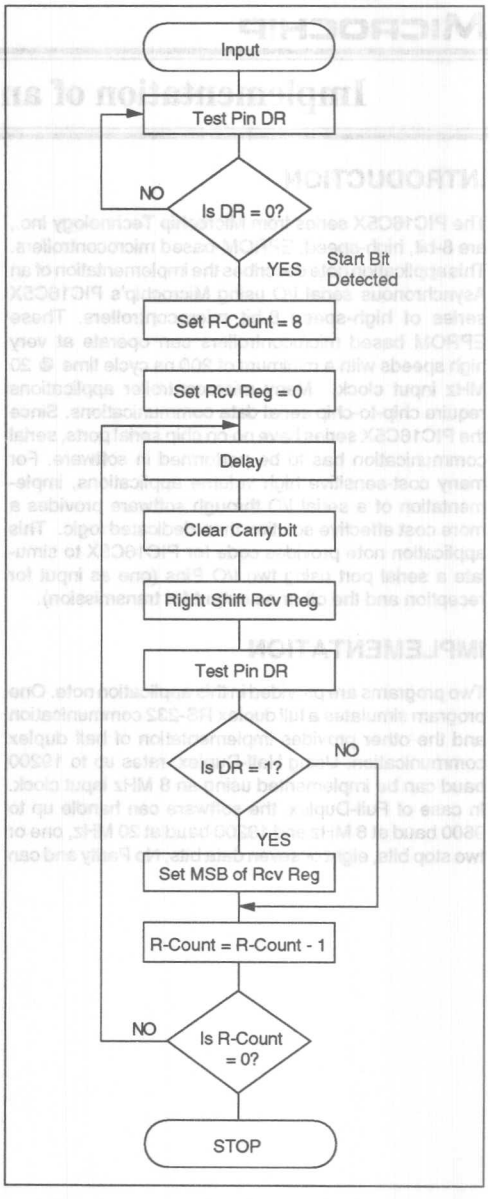


FIGURE 2 - RECEPTION FLOW CHART



© 1994 Microchip Technology Inc.

FIGURE 3 - TRANSMIT ASSEMBLY CODE (CORRESPONDING TO FIGURE 1)

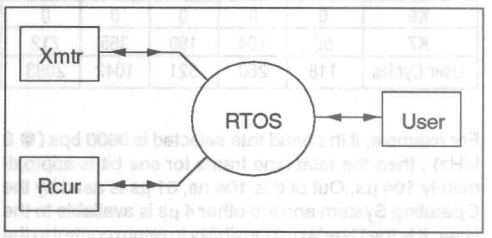
```
;********************  Transmitter*****************************
Xmtr    movlw  8            ; Assume XmtReg contains data to be Xmted
        movwf  XCount       ; 8 data bits
        bcf    Port_A,DX    ; Send Start Bit
X_next  call   Delay        ; Delay for B/2 Seconds
        rrf    XmtReg
        btfsc  STATUS,CARRY  ; Test the bit to be transmitted
        bsf    Port_A,DX    ; Bit is a one
        btfss  STATUS,CARRY
        bcf    Port_A,DX    ; Bit is zero
        decfsz Count        ; If count = 0, then transmit a stop bit
        goto   X_next       ; transmit next bit
;
X_Stop  call   Delay
        bsf    Port_A,DX     ; Send Stop Bit
X_Over  goto   X_Over
```

FIGURE 4 - RECEIVE ASSEMBLY CODE (CORRESPONDING TO FIGURE 2)

```
;***************    Receiver  *********************
;
Rcvr    btfsc  Port_A,DR    ; Test for Start Bit
        goto   Rcvr         ; Start Bit not found
        movlw  8            ; Start Bit Detected
        movwf  RCount       ; 8 Data Bits
        clrf   RcvReg       ; Receive Data Register
R_next  call   Delay        ; Delay for B/2 Seconds, B=Time duration of 1
Bit
        bcf    STATUS,CARRY ; Clear CARRY bit
        rrf    RcvReg       ; to set if MSB first or LSB first
        btfsc  Port_A,DR    ; Is the bit a zero or one ?
        bsf    RcvReg,MSB   ; Bit is a one
        call   Delay
        decfsz RCount
        goto   R_next
R_Over  goto   R_Over       ; Reception done
```

The software is organized such that the communication software acts as a Real Time Operating System (RTOS) which gives control to the User routine for a certain time interval. After this predetermined time slot, the user must give back the control to the Operating System. This is true only in the case of full-duplex implementation. Timing considerations are such that the user gets control for approximately half the time of the bit rate and the rest of the half time is used up by the Operating System (and software delays). Please refer to Table 1 for the delay constants and the time the User gets at 8 MHz input clock. Delay constants and the time that the User gets at 20 MHz and 4 MHz input clock speeds are given in the source code listing of the full duplex routine. At frequencies other than 4, 8, or 20 MHz, the delay constants and the time the user gets can be computed from the equations given in Figure 6.

FIGURE 5 - FULL DUPLEX BLOCK DIAGRAM



© 1994 Microchip Technology Inc.

FIGURE 6 - EQUATIONS FOR DELAY CONSTANTS

```
Baud_Cycles = Clkout/Baud ;
User_time = Baud_Cycles* (float) 0.5 ;
K0 = (1.25*Baud_Cycles - 2.0*User_time - 89)/3.0    ; IF (K0 < 0)
  {
  K0 = 0.0 ;
  User_time = 0.50* (1.25*Baud_Cycles - 89.0) ;
  }
K1 = (1.25*Baud_Cycles-18 - User_time - 59.0 - 3.K0)/3.0 ;
K2 = (Baud_Cycles - User_time - 41.0 - 3.K0)/3.0 ;
K3 = (Baud_Cycles - User_time - 61.0 - 3.K0)/3.0 ;
K4 = (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 ;
K5 = (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 +1.0 ;
K6 = 0.0 ;
K7 = (1.25*Baud_Cycles - User_time - 39.0 - 3.K0)/3.0 ;
```

TABLE 1 - DELAY CONSTANTS AT 8 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	-	0	5	39	109
K1	-	39	80	150	288
K2	-	27	51	86	155
K3	-	21	44	80	148
K4	-	23	46	82	150
K5	-	24	47	83	151
K6	-	0	0	0	0
K7	-	45	86	156	295
User Cycles	-	86	208	416	832

TABLE 2 - DELAY CONSTANTS AT 20 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	0	13	57	143	317
K1	49	98	184	358	705
K2	34	60	103	191	364
K3	27	53	96	184	357
K4	29	55	98	186	359
K5	30	56	99	187	360
K6	0	0	0	0	0
K7	56	104	190	365	712
User Cycles	118	260	521	1042	2083

For example, if the baud rate selected is 9600 bps (@ 8 MHz) , then the total time frame for one bit is approximately 104 µs. Out of this 104 µs, 61 µs is used by the Operating System and the other 4 µs is available to the User. It is the User's responsibility to return control to the Operating System exactly after the time specified in Table 1. For very accurate timing (with resolution up to one clock cycle) the User may set up the RTCC timer with the Prescaler option for calculating the real time. With RTCC set to increment on internal CLKOUT (500 ns @ 8 MHz CLKIN) and the prescaler assigned to it, very accurate and long timing delay loops may be assigned. This method of attaining accurate delay loops is not used in the RS232 code (RTOS), so that the RTCC is available to the User for other important functions. If the RTCC is not used for other functions, the User may modify the code to replace the software delay loops, by counting the RTCC. For an example of using this method of counting exact timing delays, refer to the "User" routine in Full-Duplex code (Appendix B).

The software uses minimal processor resources. Only six data RAM locations (File Registers) are used. The RTOS uses one level of stack, but it is freed once the control is given back to the user. The watchdog timer and RTCC are not used. The user should clear the watchdog timer at regular intervals, if the WDT is enabled.

The usage of the program is described below. The user should branch to location "Op_Sys" exactly after the time specified in Table 1 or as computed from Equations in Figure 6. Whereas, the transmission is totally under User control, the Reception is under the control of the Operating System. As long as the user does not set the X_flag, no transmission occurs. On the other hand the Operating System is constantly looking for a start bit and the user should not modify either R_done flag or RcvReg.

TRANSMISSION

Transmit Data is output on DX pin (Bit 0 of Port_A). In the user routine, the user should load the data to be transmitted in the XmtReg and Set the X_flag (bsf FlagRX,X_flag). This flag gets cleared after the transmission. The user should check this flag (X_flag) to see if transmission is in progress. Modifying XmtReg when X_flag is set will cause erroneous data to be transmitted.

RECEPTION

Data is received on pin DR (Bit 1 of Port_A). The User should constantly check the "R_done" flag to see if reception is over. If the reception is in progress, R_flag is set. If the reception is over, "R_done" flag is set to 1. The "R_done" flag gets reset to zero when a next start bit is detected. The user should constantly check the R_done flag, and if SET, then the received word is in Register "RcvReg". This register gets cleared when a new start bit is detected. It is recommended that the receive register RcvReg be copied to another register after the R_done flag is set. The R_done flag also gets cleared when the next start bit is detected.

The user may modify the code to implement an N deep buffer (limited to the number of Data RAM locations available) for receive. Also, if receiving at high speeds, and if the N deep buffer is full, an XOFF signal (HEX 13) may be transmitted. When ready to receive more data, an XON signal (HEX 11) should be transmitted.

SUMMARY

PIC16C5X family of microcontrollers allow users to implement half or full duplex RS-232 communication.

© 1994 Microchip Technology Inc.

Author: Amar Palacherla
Logic Products Division

© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  C:\AP-NOTES\   6-24-1994  8:56:3                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;                        RS-232 Communication With PIC16C54
                     0002 ;
                     0003 ;         Half Duplex Asynchronous Communication
                     0004 ;
                     0005 ;         This program has been tested at Bauds from 1200 to 19200 Baud
                     0006 ;         ( @ 8,16,20 Mhz CLKIN )
                     0007 ;
                     0008 ;        As a test, this program will echo back the data that has been
                     0009 ;        received.
                     0010 ;
                     0011 ;
                     0012          LIST            P=16C54, T=ON
                     0013      INCLUDE         "PICREG.H"
                     0001 ;*************************  PIC16C5X Header    *************************
01FF                 0002 PIC54    equ     1FFH    ; Define Reset Vectors
01FF                 0003 PIC55    equ     1FFH
03FF                 0004 PIC56    equ     3FFH
07FF                 0005 PIC57    equ     7FFH
                     0006 ;
0001                 0007 RTCC     equ     1
0002                 0008 PC       equ     2
0003                 0009 STATUS   equ     3       ; F3 Reg is STATUS Reg.
0004                 0010 FSR      equ     4
                     0011 ;
0005                 0012 Port_A   equ     5
0006                 0013 Port_B   equ     6       ; I/O Port Assignments
0007                 0014 Port_C   equ     7
                     0015 ;
                     0016 ;*******************************************************************************
                     0017 ;
                     0018 ;                        ; STATUS REG. Bits
0000                 0019 CARRY    equ     0       ; Carry Bit is Bit.0 of F3
0000                 0020 C        equ     0
0001                 0021 DCARRY   equ     1
0001                 0022 DC       equ     1
0002                 0023 Z_bit    equ     2       ; Bit 2 of F3 is Zero Bit
0002                 0024 Z        equ     2
0003                 0025 P_DOWN   equ     3
0003                 0026 PD       equ     3
0004                 0027 T_OUT    equ     4
0004                 0028 TO       equ     4
0005                 0029 PA0      equ     5
```


© 1994 Microchip Technology Inc.

```
0006                0030 PA1     equ     6
0007                0031 PA2     equ     7
                    0032 ;*****************************************************
0001                0033 Same    equ     1
0000                0034 W       equ     0
                    0035 ;*****************************************************
0000                0036 LSB     equ     0
0007                0037 MSB     equ     7
                    0038 ;
0001                0039 TRUE    equ     1
0001                0040 YES     equ     1
0000                0041 FALSE   equ     0
0000                0042 NO      equ     0
                    0043 ;
                    0044 ;*************************************************************************
                    0045
                    0013
                    0014 ;*****************  Communication Parameters   **************************
                    0015 ;
0001                0016 X_MODE  equ     1       ; If ( X_MODE==1) Then transmit LSB first
                    0017 ;                       ; if ( X_MODE==0) Then transmit MSB first ( CODEC like )
0001                0018 R_MODE  equ     1       ; If ( R_MODE==1) Then receive LSB first
                    0019 ;                       ; if ( X_MODE==0) Then receive MSB first ( CODEC like )
0001                0020 X_Nbit  equ     1       ; if (X_Nbit==1) # of data bits ( Transmission ) is 8 else 7
0001                0021 R_Nbit  equ     1       ; if (R_Nbit==1) # of data bits ( Reception ) is 8 else 7
                    0022 ;
0000                0023 Sbit2   equ     0       ; if Sbit2 = 0 then 1 Stop Bit else 2 Stop Bits
                    0024 ;
                    0025 ;*************************************************************************
0005                0026 X_flag  equ     PA0     ; Bit 5 of F3 ( PA0 )
0006                0027 R_flag  equ     PA1     ; Bit 6 of F3 ( PA1 )
                    0028 ;
0000                0029 DX      equ     0       ; Transmit Pin ( Bit 0 of Port A )
0001                0030 DR      equ     1       ; Reciive Pin ( Bit 1 of Port A )
                    0031 ;
                    0032 ;
0044                0033 BAUD_1  equ     .68     ; 3+3X = CLKOUT/Baud
0043                0034 BAUD_2  equ     .67     ; 6+3X = CLKOUT/Baud
0022                0035 BAUD_3  equ     .34     ; 3+3X = 0.5*CLKOUT/Baud
0056                0036 BAUD_4  equ     .86     ; 3+3X = 1.25*CLKOUT/Baud
0042                0037 BAUD_X  equ     .66     ; 11+3X = CLKOUT/Baud
0042                0038 BAUD_Y  equ     .66     ; 9 +3X = CLKOUT/Baud
                    0039 ;
                    0040 ;***********************  Data RAM Assignments  **********************
                    0041 ;
                    0042          ORG     08H     ; Dummy Origin
                    0043 ;
0008  0001          0044 RcvReg  RES     1       ; Data received
```

© 1994 Microchip Technology Inc.

```
0009  0001          0045 XmtReg  RES     1              ; Data to be transmitted
000A  0001          0046 Count   RES     1              ; Counter for #of Bits Transmitted
000B  0001          0047 DlyCnt  RES     1
                    0048 ;*************************************************************************
                    0049 ;
                    0050         ORG     0
                    0051 ;
0000 0068           0052 Talk    clrf    RcvReg         ; Clear all bits of RcvReg
0001 0625           0053         btfsc   Port_A,DR      ; check for a Start Bit
0002 0A30           0054         goto    User           ; delay for 104/2 uS
0003 0923           0055         call    Delay4         ; delay for 104+104/4
                    0056 ;*************************************************************
                    0057 ;       Receiver
                    0058 ;
                    0059 Rcvr
                    0060         IF      R_Nbit
0004 0C08           0061         movlw   8              ; 8 Data bits
                    0062         ELSE
                    0063         movlw   7              ; 7 data bits
                    0064         ENDIF
                    0065 ;
0005 002A           0066         movwf   Count
0006 0403           0067 R_next  bcf     STATUS,CARRY
                    0068         IF      R_MODE
0007 0328           0069         rrf     RcvReg,Same    ; to set if MSB first or LSB first
                    0070         ELSE
                    0071         rlf     RcvReg,Same
                    0072         ENDIF
0008 0625           0073         btfsc   Port_A,DR
                    0074 ;
                    0075         IF      R_MODE
                    0076           IF      R_Nbit
0009 05E8           0077           bsf     RcvReg,MSB     ; Conditional Assembly
                    0078           ELSE
                    0079           bsf     RcvReg,MSB-1
                    0080           ENDIF
                    0081         ELSE
                    0082         bsf     RcvReg,LSB
                    0083         ENDIF
                    0084 ;
000A 091F           0085         call    DelayY
000B 02EA           0086         decfsz  Count,Same
000C 0A06           0087         goto    R_next
                    0088 ;*************************************************
000D 0208           0089 R_over  movf    RcvReg,0       ; Send back What is Just Received
000E 0029           0090         movwf   XmtReg
                    0091 ;*************************************************
                    0092 ;       Transmitter
```


© 1994 Microchip Technology Inc.

```
                    0093 ;
                    0094 Xmtr
                    0095          IF      X_Nbit
000F 0C08           0096          movlw   8
                    0097          ELSE
                    0098          movlw   7
                    0099          ENDIF
0010 002A           0100          movwf   Count
                    0101 ;
                    0102          IF      X_MODE
                    0103          ELSE
                    0104            IF    X_Nbit
                    0105            ELSE
                    0106            rlf   XmtReg,Same
                    0107            ENDIF
                    0108          ENDIF
                    0109 ;
0011 0405           0110          bcf     Port_A,DX       ; Send Start Bit
0012 0925           0111          call    Delay1
0013 0403           0112 X_next   bcf     STATUS,CARRY
                    0113 ;
                    0114          IF      X_MODE
0014 0329           0115          rrf     XmtReg,Same     ; Conditional Assembly
                    0116          ELSE                    ; to set if MSB first or LSB first
                    0117          rlf     XmtReg,Same
                    0118          ENDIF
                    0119 ;
0015 0603           0120          btfsc   STATUS,CARRY
0016 0505           0121          bsf     Port_A,DX
0017 0703           0122          btfss   STATUS,CARRY
0018 0405           0123          bcf     Port_A,DX
0019 0921           0124          call    DelayX
001A 02EA           0125          decfsz  Count,Same
001B 0A13           0126          goto    X_next
001C 0505           0127          bsf     Port_A,DX       ; Send Stop Bit
001D 0925           0128          call    Delay1
                    0129 ;
                    0130          IF      Sbit2
                    0131          bsf     Port_A,DX
                    0132          call    Delay1
                    0133          ENDIF
                    0134 ;
001E 0A00           0135          goto    Talk            ; Back To Reception & Transmision
                    0136 ;
                    0137 ;   End of Transmission
                    0138 ;
001F 0C42           0139 DelayY   movlw   BAUD_Y
0020 0A28           0140          goto    save
```

© 1994 Microchip Technology Inc.

```
0021 0C42            0141 DelayX  movlw   BAUD_X
0022 0A28            0142         goto    save
0023 0C56            0143 Delay4  movlw   BAUD_4
0024 0A28            0144         goto    save
0025 0C44            0145 Delay1  movlw   BAUD_1          ; 104 uS for 9600 baud
0026 0A28            0146         goto    save
0027 0C43            0147 Delay2  movlw   BAUD_2
0028 002B            0148 save    movwf   DlyCnt
0029 02EB            0149 redo_1  decfsz  DlyCnt,Same
002A 0A29            0150         goto    redo_1
002B 0800            0151         retlw   0
                     0152 ;
002C 0C0E            0153 main    movlw   0EH             ; Bit 0 of Port A is Output
002D 0005            0154         tris    Port_A          ; Set Port_A.0 as output ( DX )
002E 0525            0155         bsf     Port_A,DR
                     0156 ;
002F 0A00            0157         goto    Talk
                     0158 ;
                     0159 ;
0030 0C22            0160 User    movlw   BAUD_3
0031 002B            0161         movwf   DlyCnt
0032 02EB            0162 redo_2  decfsz  DlyCnt,Same
0033 0A32            0163         goto    redo_2
0034 0A00            0164         goto    Talk            ; Loop Until Start Bit Found
                     0165 ;
                     0166 ;
                     0167         ORG     PIC54
01FF 0A2C            0168         goto    main
                     0169 ;
                     0170         END
                     0171
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX-----
0040 : -------- -------- -------- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- --------X

All other memory blocks unused.
Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  C:\AP-NOTES\    6-24-1994  9:8:43                PAGE  1
RS232 Communication Using PIC16C54

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;*******************************************************************************
                     0002         TITLE   "RS232 Communication Using PIC16C54"
                     0003 ;
                     0004 ;    Comments :
                     0005 ;                (1) Full Duplex
                     0006 ;                (2) Tested from 1200 to 9600 Baud( @ 8 Mhz )
                     0007 ;                (3) Tested from 1200 to 19200 Baud(@ 16 & 20 Mhz)
                     0008 ;
                     0009 ;        The User gets a total time as specified by the User Cycles
                     0010 ;        in the table ( or from equations ). The user routine has to
                     0011 ;        exactly use up this amount of time. After this time the User
                     0012 ;        routine has to give up the control to the Operating System.
                     0013 ;        If less than 52 uS is used, then the user should wait in a
                     0014 ;        delay loop, until exactly 52 uS.
                     0015 ;
                     0016 ;        Transmission :
                     0017 ;                Transmit Data is output on DX pin (Bit DX of Port_A).
                     0018 ;                In the user routine, the user should load the
                     0019 ;                data to be transmitted in the XmtReg and Set the
                     0020 ;                X_flag ( bsf FlagRX,X_flag ). This flag gets cleared
                     0021 ;                after the transmission.
                     0022 ;
                     0023 ;        Reception :
                     0024 ;                Data is received on pin DR ( bit DR of Port_A ).
                     0025 ;                The User should constantly check the "R_done" flag
                     0026 ;                to see if reception is over. If the reception is
                     0027 ;                in progress, R_flag is set to 1.
                     0028 ;                If the reception is over, "R_done" flag is set to 1.
                     0029 ;                The "R_done" flag gets reset to zero when a next start
                     0030 ;                bit is detected. So, the user should constantly check
                     0031 ;                the R_done flag, and if SET, then the received word
                     0032 ;                is in Register "RcvReg". This register gets cleared
                     0033 ;                when a new start bit is detected.
                     0034 ;
                     0035 ;    Program Memory :
                     0036 ;       Total Program Memory Locations Used ( except initialization
                     0037 ;       in "main" & User routine ) = 132 locations.
                     0038 ;
                     0039 ;    Data Memory :
                     0040 ;       Total Data memory locations (file registers used) = 6
                     0041 ;                2 File registers to hold Xmt Data & Rcv Data
                     0042 ;                1 File registers for Xmt/Rcv flag test bits
```

© 1994 Microchip Technology Inc.

```
                        0043 ;              3 File registers for delay count & scratch pad
                        0044 ;
                        0045 ;    Stack :
                        0046 ;       Only one level of stack is used in the Operating System/RS232
                        0047 ;       routine. But this is freed as soon as the program returns to the
                        0048 ;       user routine.
                        0049 ;
                        0050 ;    RTCC :     Not Used
                        0051 ;    WDT  :     Not Used
                        0052 ;
                        0053          LIST              P=16C54, T=ON
                        0054      INCLUDE          "PICREG.H"
                        0001 ;************************  PIC16C5X Header    ************************
01FF                    0002 PIC54    equ     1FFH    ; Define Reset Vectors
01FF                    0003 PIC55    equ     1FFH
03FF                    0004 PIC56    equ     3FFH
07FF                    0005 PIC57    equ     7FFH
                        0006 ;
0001                    0007 RTCC     equ     1
0002                    0008 PC       equ     2
0003                    0009 STATUS   equ     3       ; F3 Reg is STATUS Reg.
0004                    0010 FSR      equ     4
                        0011 ;
0005                    0012 Port_A   equ     5
0006                    0013 Port_B   equ     6       ; I/O Port Assignments
0007                    0014 Port_C   equ     7
                        0015 ;
                        0016 ;*****************************************************************************
                        0017 ;
                        0018 ;                        ; STATUS REG. Bits
0000                    0019 CARRY    equ     0       ; Carry Bit is Bit.0 of F3
0000                    0020 C        equ     0
0001                    0021 DCARRY   equ     1
0001                    0022 DC       equ     1
0002                    0023 Z_bit    equ     2       ; Bit 2 of F3 is Zero Bit
0002                    0024 Z        equ     2
0003                    0025 P_DOWN   equ     3
0003                    0026 PD       equ     3
0004                    0027 T_OUT    equ     4
0004                    0028 TO       equ     4
0005                    0029 PA0      equ     5
0006                    0030 PA1      equ     6
0007                    0031 PA2      equ     7
                        0032 ;
0001                    0033 Same     equ     1
0000                    0034 W        equ     0
                        0035 ;
0000                    0036 LSB      equ     0
```


© 1994 Microchip Technology Inc.

```
0007                0037 MSB     equ     7
                    0038 ;
0001                0039 TRUE    equ     1
0001                0040 YES     equ     1
0000                0041 FALSE   equ     0
0000                0042 NO      equ     0
                    0043 ;
                    0044 ;*****************************************************************************
                    0045
                    0054 ;-----------------------------------------------------------------------------
                    0055     INCLUDE         "RS232.H"
                    0001 ;*****************************************************************************
                    0002 ;                       RS232 Communication Parameters
                    0003 ;
                    0004 ;
0001                0005 X_MODE  equ     1       ; If ( X_MODE==1) Then transmit LSB first
                    0006 ;                         if ( X_MODE==0) Then transmit MSB first ( CODEC like )
0001                0007 R_MODE  equ     1       ; If ( R_MODE==1) Then receive LSB first
                    0008 ;                         if ( R_MODE==0) Then receive MSB first ( CODEC like )
0001                0009 X_Nbit  equ     1       ; if ( X_Nbit==1) # of data bits ( Transmission ) is 8 else 7
0001                0010 R_Nbit  equ     1       ; if ( R_Nbit==1) # of data bits ( Reception ) is 8 else 7
                    0011 ;
0000                0012 SB2     equ     0       ; if SB2 = 0 then 1 Stop Bit
                    0013 ;                       ; if SB2 = 1 then 2 Stop Bit
                    0014 ;*****************************************************************************
                    0015 ;       Transmit & Receive Test Bit Assignments
                    0016 ;
0000                0017 X_flag  equ     0       ; Bit 0 of FlagRX
0002                0018 R_flag  equ     2       ; Bit 1 of FlagRX
0003                0019 S_flag  equ     3       ; Bit 2 of FlagRX
0004                0020 BitXsb  equ     4       ; Bit 3 of FlagRX
0005                0021 A_flag  equ     5
0006                0022 S_bit   equ     6       ; Xmt Stop Bit Flag( for 2/1 Stop bits )
                    0023 ;
0001                0024 R_done  equ     1       ; When Reception complete, this bit is SET
0000                0025 X_done  equ     X_flag  ; When Xmission complete, this bit is Cleared
                    0026 ;
0000                0027 DX      equ     0       ; Transmit Pin ( Bit 0 of Port A )
0001                0028 DR      equ     1       ; Reciive Pin ( Bit 1 of Port A )
                    0029 ;
                    0030 ;*********************  Data RAM Assignments  *********************
                    0031 ;
                    0032         ORG     08H     ; Dummy Origin
                    0033 ;
0008  0001          0034 RcvReg  RES     1       ; Data received
0009  0001          0035 XmtReg  RES     1       ; Data to be transmitted
000A  0001          0036 Xcount  RES     1       ; Counter for #of Bits Transmitted
000B  0001          0037 Rcount  RES     1       ; Counter for #of Bits to be Received
```

© 1994 Microchip Technology Inc.

```
000C  0001          0038 DlyCnt  RES     1       ; Counter for Delay constant
000D  0001          0039 FlagRX  RES     1       ; Transmit & Receive test flag hold register
                    0040 ;_____________________________________________________________
                    0041 ;    Constants       19200   9600    4800    2400    1200
                    0042 ;  ( @ 20 Mhz )
                    0043 ;_____________________________________________________________
                    0044 ;        K0            0      13      57     143     317*
                    0045 ;        K1           49      98     184     358*    705*
                    0046 ;        K2           34      60     103     191     364*
                    0047 ;        K3           27      53      96     184     357*
                    0048 ;        K4           29      55      98     186     359*
                    0049 ;        K5           30      56      99     187     360*
                    0050 ;        K6            0       0       0       0       0
                    0051 ;        K7           56     104     190     365*    712*
                    0052 ;
                    0053 ;    User Cycles     118     260     521    1042    2083
                    0054 ; ************************************************************
                    0055 ;
                    0056 ;
                    0057 ;_____________________________________________________________
                    0058 ;    Constants       19200   9600    4800    2400    1200
                    0059 ;  ( @ 8 Mhz )
                    0060 ;_____________________________________________________________
                    0061 ;        K0            -       0       5      39     109
                    0062 ;        K1            -      39      80     150     288*
                    0063 ;        K2            -      27      51      86     155
                    0064 ;        K3            -      21      44      80     148
                    0065 ;        K4            -      23      46      82     150
                    0066 ;        K5            -      24      47      83     151
                    0067 ;        K6            -       0       0       0       0
                    0068 ;        K7            -      45      86     156     295*
                    0069 ;
                    0070 ;    User_Cycles       -      86     208     416     832
                    0071 ; ************************************************************
                    0072 ;
                    0073 ;_____________________________________________________________
                    0074 ;    Constants       19200   9600    4800    2400    1200
                    0075 ;  ( @ 4 Mhz )
                    0076 ;_____________________________________________________________
                    0077 ;        K0            -       -       0       5      39
                    0078 ;        K1            -       -      39      80     150
                    0079 ;        K2            -       -      27      51      86
                    0080 ;        K3            -       -      21      44      80
                    0081 ;        K4            -       -      23      46      82
                    0082 ;        K5            -       -      24      47      83
                    0083 ;        K6            -       -       0       0       0
                    0084 ;        K7            -       -      45      86     156
                    0085 ;
```


© 1994 Microchip Technology Inc.

```
                    0086 ;    User_Cycles      -      -      86     208      416
                    0087 ; *************************************************************
                    0088
                    0089 ;
                    0090 ; The constants marked " * " are >255. To implement these constants
                    0091 ; in delay loops, the delay loop should be broken into 2 or more loops.
                    0092 ; For example, 357 = 255+102. So 2 delay loops, one with 255 and
                    0093 ; the other with 102 may be used.
                    0094 ;*****************************************************************************
                    0095 ;       Set Delay Constants for 9600 Baud @ CLKIN = 8 Mhz
                    0096 ;
0000                0097 K0      EQU     .0
0027                0098 K1      EQU     .39
001B                0099 K2      EQU     .27
0015                0100 K3      EQU     .21
0017                0101 K4      EQU     .23
0018                0102 K5      EQU     .24
0000                0103 K6      EQU     .0
002D                0104 K7      EQU     .45
                    0105 ;
                    0106 ;*****************************************************************************
                    0107
                    0055
                    0056 ;
                    0057         ORG     0
                    0058 ;*****************************************************************************
                    0059 ;
0000 0C01           0060 Delay   movlw   K0+1
0001 002C           0061         movwf   DlyCnt          ; Total Delay = 3K+6
0002 02EC           0062 redo    decfsz  DlyCnt,Same
0003 0A02           0063         goto    redo
0004 0800           0064         retlw   0
                    0065 ;
0005 002C           0066 Delay1  movwf   DlyCnt
0006 02EC           0067 redo_1  decfsz  DlyCnt,Same     ;
0007 0A06           0068         goto    redo_1
0008 0A8D           0069         goto    User
                    0070 ;
0009 002C           0071 Delay2  movwf   DlyCnt
000A 02EC           0072 redo_2  decfsz  DlyCnt,Same     ; Delay =   = 260 Cycles
000B 0A0A           0073         goto    redo_2
000C 0A67           0074         goto    User_1
                    0075 ;
000D 0625           0076 R_strt  btfsc   Port_A,DR       ; check for a Start Bit
000E 0A17           0077         goto    ShellY          ; delay for 104/2 uS
000F 042D           0078         bcf     FlagRX,R_done   ; Reset Receive done flag
0010 054D           0079         bsf     FlagRX,R_flag   ; Set flag for Reception in Progress
0011 078D           0080         btfss   FlagRX,BitXsb
```

© 1994 Microchip Technology Inc.

```
0012 05AD          0081          bsf     FlagRX,A_flag   ; A_flag is for start bit detected in R_strt
0013 0068          0082          clrf    RcvReg          ; Clear all bits of RcvReg
                   0083          IF      R_Nbit
0014 0C08          0084          movlw   8               ; 8 Data bits
                   0085          ELSE
                   0086          movlw   7               ; 7 data bits
                   0087          ENDIF
0015 002B          0088          movwf   Rcount
0016 0A78          0089          goto    Shell           ; delay for 104+104/4
                   0090 ;
0017 078D          0091 ShellY   btfss   FlagRX,BitXsb
0018 0A78          0092          goto    Shell
0019 054D          0093          bsf     FlagRX,R_flag
001A 0A78          0094          goto    Shell
                   0095 ;
001B 0403          0096 R_next   bcf     STATUS,CARRY
                   0097          IF      R_MODE
001C 0328          0098          rrf     RcvReg,Same     ; to set if MSB first or LSB first
                   0099          ELSE
                   0100          rlf     RcvReg,Same
                   0101          ENDIF
001D 0625          0102          btfsc   Port_A,DR
                   0103          IF      R_MODE
                   0104            IF      R_Nbit
001E 05E8          0105            bsf     RcvReg,MSB      ; Conditional Assembly
                   0106            ELSE
                   0107            bsf     RcvReg,MSB-1
                   0108            ENDIF
                   0109          ELSE
                   0110          bsf     RcvReg,LSB
                   0111          ENDIF
001F 02EB          0112          decfsz  Rcount,Same
0020 0A78          0113          goto    Shell
0021 044D          0114          bcf     FlagRX,R_flag
0022 056D          0115          bsf     FlagRX,S_flag
0023 052D          0116          bsf     FlagRX,R_done
0024 0A78          0117          goto    Shell
                   0118 ;
                   0119 ;        Reception Done
                   0120 ;
0025 0405          0121 X_strt   bcf     Port_A,DX       ; Send Start Bit
                   0122          IF      X_Nbit
0026 0C08          0123          movlw   8
                   0124          ELSE
                   0125          movlw   7
                   0126          ENDIF
0027 002A          0127          movwf   Xcount
                   0128          IF      X_MODE
```


© 1994 Microchip Technology Inc.

```
                        0129          ELSE
                        0130            IF    X_Nbit
                        0131            ELSE
                        0132            rlf   XmtReg,Same
                        0133            ENDIF
                        0134          ENDIF
0028 0A50               0135          goto    X_SB
                        0136 ;
0029 0403               0137 X_next   bcf     STATUS,CARRY
                        0138          IF      X_MODE
002A 0329               0139          rrf     XmtReg,Same     ; Conditional Assembly
                        0140          ELSE                    ; to set if MSB first or LSB first
                        0141          rlf     XmtReg,Same
                        0142          ENDIF
002B 0603               0143          btfsc   STATUS,CARRY
002C 0505               0144          bsf     Port_A,DX
002D 0703               0145          btfss   STATUS,CARRY
002E 0405               0146          bcf     Port_A,DX
002F 00EA               0147          decf    Xcount,Same
0030 0A52               0148          goto    X_Data
                        0149 ;
0031 040D               0150 X_SB_1   bcf     FlagRX,X_flag   ; Xmt flag = 0 - transmission over
0032 0C09               0151          movlw   9
0033 002A               0152          movwf   Xcount
0034 0505               0153          bsf     Port_A,DX       ; Send Stop Bit
0035 0A60               0154          goto    X_Stop
                        0155 ;
0036 0505               0156 X_SB_2   bsf     Port_A,DX
0037 04CD               0157          bcf     FlagRX,S_bit
0038 0A60               0158          goto    X_Stop
                        0159 ;
                        0160 ;   End of Transmission
                        0161 ;
0039 076D               0162 R0_X0    btfss   FlagRX,S_flag
003A 0A8D               0163          goto    User
003B 046D               0164          bcf     FlagRX,S_flag
003C 0900               0165          call    Delay
003D 0C2E               0166          movlw   K7+1
003E 0A05               0167          goto    Delay1
                        0168 ;
                        0169 R1_X0
003F 0900               0170          call    Delay
0040 0C28               0171          movlw   K1+1            ; delay for 1st bit is 104+104/4
0041 002C               0172          movwf   DlyCnt
                        0173          IF      R_Nbit
0042 0C08               0174          movlw   8               ; 8 Data bits
                        0175          ELSE
                        0176          movlw   7               ; 7 data bits
```

© 1994 Microchip Technology Inc.

```
                     0177          ENDIF
0043 018B            0178          xorwf   Rcount,W
0044 0643            0179          btfsc   STATUS,Z_bit
0045 0A06            0180          goto    redo_1
0046 0C1C            0181          movlw   K2+1
0047 0A05            0182          goto    Delay1
                     0183 ;
                     0184 R1_X1                            ; same as R0_X1
0048 0C09            0185 R0_X1    movlw   9
0049 008A            0186          subwf   Xcount,W
004A 0643            0187          btfsc   STATUS,Z_bit
004B 0A25            0188          goto    X_strt
004C 022A            0189          movf    Xcount,Same     ; to check if All data bits Xmted
004D 0743            0190          btfss   STATUS,Z_bit
004E 0A29            0191          goto    X_next
                     0192          IF      SB2
                     0193            btfsc FlagRX,S_bit
                     0194            goto  X_SB_2
                     0195            bsf   FlagRX,S_bit
                     0196            goto  X_SB_1
                     0197          ELSE
004F 0A31            0198            goto    X_SB_1
                     0199          ENDIF
                     0200 ;
                     0201 ;
0050 0A51            0202 X_SB     goto    cycle4
0051 0A52            0203 cycle4   goto    X_Data
                     0204 ;
0052 06AD            0205 X_Data   btfsc   FlagRX,A_flag
0053 0A59            0206          goto    SbDly
0054 068D            0207          btfsc   FlagRX,BitXsb
0055 0A5D            0208          goto    ABC
0056 0900            0209          call    Delay
0057 0C16            0210          movlw   K3+1
0058 0A09            0211          goto    Delay2
                     0212 ;
0059 04AD            0213 SbDly    bcf     FlagRX,A_flag
005A 0900            0214          call    Delay
005B 0C18            0215          movlw   K4+1
005C 0A09            0216          goto    Delay2
                     0217 ;
005D 048D            0218 ABC      bcf     FlagRX,BitXsb
005E 0900            0219          call    Delay
005F 0A67            0220          goto    User_1
                     0221 ;
                     0222 X_Stop
0060 06AD            0223          btfsc   FlagRX,A_flag
0061 0A59            0224          goto    SbDly
```


© 1994 Microchip Technology Inc.

```
0062 068D          0225          btfsc   FlagRX,BitXsb
0063 0A5D          0226          goto    ABC
0064 0900          0227          call    Delay
0065 0C19          0228          movlw   K5+1
0066 0A09          0229          goto    Delay2
                   0230 ;
0067 064D          0231 User_1   btfsc   FlagRX,R_flag
0068 0A77          0232          goto    Sync_1          ; Reception already in progress
0069 066D          0233          btfsc   FlagRX,S_flag
006A 0A74          0234          goto    Sync_3
006B 0625          0235          btfsc   Port_A,DR       ; check for a Start Bit
006C 0A77          0236          goto    Sync_2          ; No Start Bit - goto User routine
006D 042D          0237          bcf     FlagRX,R_done   ; Reset Receive done flag
006E 044D          0238          bcf     FlagRX,R_flag
006F 058D          0239          bsf     FlagRX,BitXsb   ; Set flag for Reception in Progress
0070 0068          0240          clrf    RcvReg          ; Clear all bits of RcvReg
                   0241          IF      R_Nbit
0071 0C08          0242          movlw   8               ; 8 Data bits
                   0243          ELSE
                   0244          movlw   7               ; 7 data bits
                   0245          ENDIF
0072 002B          0246          movwf   Rcount
0073 0A8D          0247          goto    User
                   0248 ;
0074 046D          0249 Sync_3   bcf     FlagRX,S_flag
0075 0C01          0250          movlw   K6+1
0076 0A05          0251          goto    Delay1
                   0252 ;
                   0253 Sync_1
0077 0A8D          0254 Sync_2   goto    User
                   0255 ;
                   0256 ;*************************************************************************
                   0257 ;
0078 064D          0258 Shell    btfsc   FlagRX,R_flag
0079 0A7D          0259          goto    Chek_X
007A 060D          0260          btfsc   FlagRX,X_flag
007B 0A48          0261          goto    R0_X1
007C 0A39          0262          goto    R0_X0           ; Case for R0_X0
007D 060D          0263 Chek_X   btfsc   FlagRX,X_flag
007E 0A48          0264          goto    R1_X1
007F 0A3F          0265          goto    R1_X0
                   0266 ;
                   0267 ;
                   0268 ;*************************************************************************
                   0269 ;        Operating System
                   0270 ;  The User routine after time = B/2, should branch Here
                   0271 ;
0080 074D          0272 Op_Sys   btfss   FlagRX,R_flag
```

© 1994 Microchip Technology Inc.

```
0081 0A0D          0273          goto    R_strt
0082 0A1B          0274          goto    R_next
                   0275 ;
                   0276 ;*****************************************************************************
                   0277 ;
0083 0C0E          0278 main     movlw   0EH             ; Bit 0 of Port A is Output
0084 0005          0279          tris    Port_A          ; Set Port_A.0 as output ( DX )
                   0280 ;                                  & Port_A.1 is input ( DR )
0085 0505          0281          bsf     Port_A,DX
0086 0C09          0282          movlw   9
0087 002A          0283          movwf   Xcount          ; If Xcount == 9, Then send start bit
0088 006D          0284          clrf    FlagRX          ; Clear All flag bits.
                   0285          IF      SB2
                   0286            bsf     FlagRX,S_bit  ; Set Xmt Stop bit flag(2 Stop Bits)
                   0287          ELSE
0089 04CD          0288            bcf     FlagRX,S_bit  ; Clear Xmt Stop bit flag
                   0289          ENDIF
008A 0C1F          0290          movlw   1FH             ; Prescaler = 4
008B 0002          0291          OPTION                  ; Set RTCC increment on internal Clock
008C 0A80          0292          goto    Op_Sys
                   0293 ;
                   0294 ;*****************************************************************************
                   0295 ;
                   0296 ;                 ******************  User Routine  ***************
                   0297 ;  The User routine should use up time exactly = User time as given
                   0298 ;  in the Constants Table ( or by Equations for constants ).
                   0299 ;  At 9600, this 86 Clock Cycles. RTCC timer is used here to count
                   0300 ;  upto 86 cycles ( From 128-86 To 0 ) by examining Bit 7 of RTCC.
                   0301 ;
0030               0302 K_user   equ     .128+.6-.86
                   0303 ;
008D 0C30          0304 User     movlw   K_user
008E 0021          0305          movwf   RTCC
008F 062D          0306          btfsc   FlagRX,R_done
0090 0A97          0307          goto    ErrChk
0091 060D          0308 SetXmt   btfsc   FlagRX,X_flag
0092 0A9C          0309          goto    Op
0093 0C41          0310          movlw   41H
0094 0029          0311          movwf   XmtReg
0095 050D          0312          bsf     FlagRX,X_flag   ; Enable Xmission
0096 0A9C          0313          goto    Op
                   0314 ;
                   0315 ErrChk
0097 0C5A          0316          movlw   "Z"
0098 0188          0317          xorwf   RcvReg,W
0099 0643          0318          btfsc   STATUS,Z_bit
009A 0A91          0319          goto    SetXmt
009B 0A9B          0320 error    goto    error           ; Received word is not "Z"
```


© 1994 Microchip Technology Inc.

```
                        0321 ;
009C 07E1               0322 Op      btfss   RTCC,MSB        ; Test for RTCC bit 7
009D 0A9C               0323         goto    Op              ; If Set, Then RTCC has incremented
009E 0A80               0324 Oflow   goto    Op_Sys          ; to 128.
                        0325 ;
                        0326 ; *******************************************************
                        0327 ;
                        0328         ORG     PIC54
01FF 0A83               0329         goto    main
                        0330
                        0331         END
                        0332
                        0333

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- -------- --------
00C0 : -------- -------- -------- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Using a PIC16C5X as a Smart I²C™ Peripheral

Author: Don Lekei - NII Norsat International Inc.

INTRODUCTION

The PIC16C5X microcontrollers from Microchip are ideally suited for use as smart peripheral devices under the control of the main processors in systems due to their low cost and high speed. They are capable of performing tasks which would simply overload a conventional microprocessor, or require considerable logic circuitry, at a cost competitive with lower mid-range PLDs. To minimize the engineering overhead of adding multiple controllers to a product, it is convenient for the auxiliary controllers to emulate standard I/O peripherals.

A common interface found in existing products is the I²C bus. This efficient, two-wire, bi-directional interface allows the designer to connect multiple devices together, with the microprocessor able to send data to and receive data from any device on the bus. This interface is found on a variety of components, such as PLLs, DACs, video controllers, and EEPROMs. If a product already contains one or more I²C devices, it is simple to add a PIC16C5X emulating a compatible component.

This application note describes the implementation of a standard slave device with multiple, bi-directional registers. A subset of the full I²C specification is supported, which can be controlled by the same software which would talk to a Microchip 24LCXX series EEPROM.

THE I²C BUS

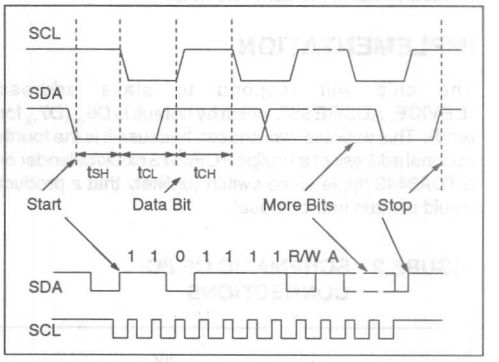
The I²C bus is a master-slave two-wire interface, consisting of a clock line (SCL) and a data line (SDA). Bi-directional communication (and in a full, multi-master system, collision detection and clock synchronization) is facilitated through the use of a wire-and (ie. active-low, passive high) connection.

The standard-mode I²C bus supports SCL clock frequency up to 100 KHz. The newly released fast-mode I²C bus supports clock rate up to 400 KHz. This application note will support 100 KHz (standard-mode) clock rate.

Each device has a unique seven bit address, which the master uses to access each individual slave device.

During normal communication, SDA is only permitted to change while SCL is low, thus providing two violation conditions (see Figure 1) which are used to signal a start condition (SDA drops while SCL is high) and a stop condition (SDA rises while SCL is high), which frame a message.

FIGURE 1 - I²C TIMING



Each byte of a transfer is 9-bits long (see timing chart in the program listing). The talker sends 8 data bits followed by a "1" bit. The listener acknowledges the receipt of the byte and permission to send the next byte by inserting a "0" bit over the trailing "1". The listener may indicate "not ready for data" by leaving the acknowledge bit as a "1".

The clock is generated by the master only. The slave device must respond to the master within the timing specifications of the I²C definition otherwise the master would be required to operate in slow mode, which most software implementations of I²C masters do not actually support. The specified (standard-mode) t_{CL} is 4.7 µs, and t_{CH} is only 4 µs, so it would be extremely difficult to achieve the timing of a hardware slave device with a conventional microcontroller.

MESSAGE FORMAT

A message is always initiated by the master, and begins with a start condition, followed by a slave address (7 MSBs) and direction bit (LSB = 1 for READ, 0 for WRITE). The addressed slave must acknowledge this byte if it is ready to communicate any data. If the slave fails to respond, the master should send a stop and retry.

If the direction bit is "0" the next byte is considered the sub-address (this is an extension to I²C used by most multi-register devices). The sub-address selects which "register" or function subsequent read or write operations will affect. Any additional bytes will be received and stored in consecutive locations until a stop is sent. If the slave is unable to process more data, it could terminate transfer by not acknowledging the last byte.

© 1993 Microchip Technology Inc.

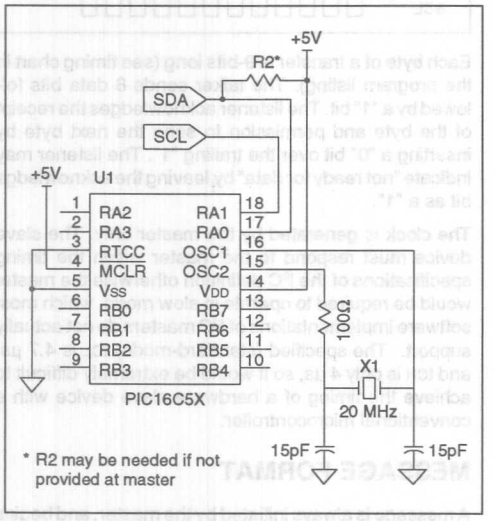
If the direction bit is "1", the slave will transfer successive bytes to the master (while the master holds the line at '1'), while the master acknowledges each byte with a "0" in the ninth bit. The master can terminate the transfer by not acknowledging the last byte, while the slave can stop the transfer by generating a stop condition.

The start address of a read operation is set by sending a write request with a sub-address only (no data bytes). For a detailed set of timing diagrams and different communication modes, consult any of the Microchip 24LCXX EEPROM specifications. This program communicates using the same formats.

IMPLEMENTATION

The chip will respond to slave address "DEVICE_ADDRESS", which by default is $D6_{16}$ ($D7_{16}$ for read). This address was chosen because it is the fourth optional address of a Philips PCF8573 clock/calender or a TDA8443 tipple video switch (unlikely that a product would contain four of those).

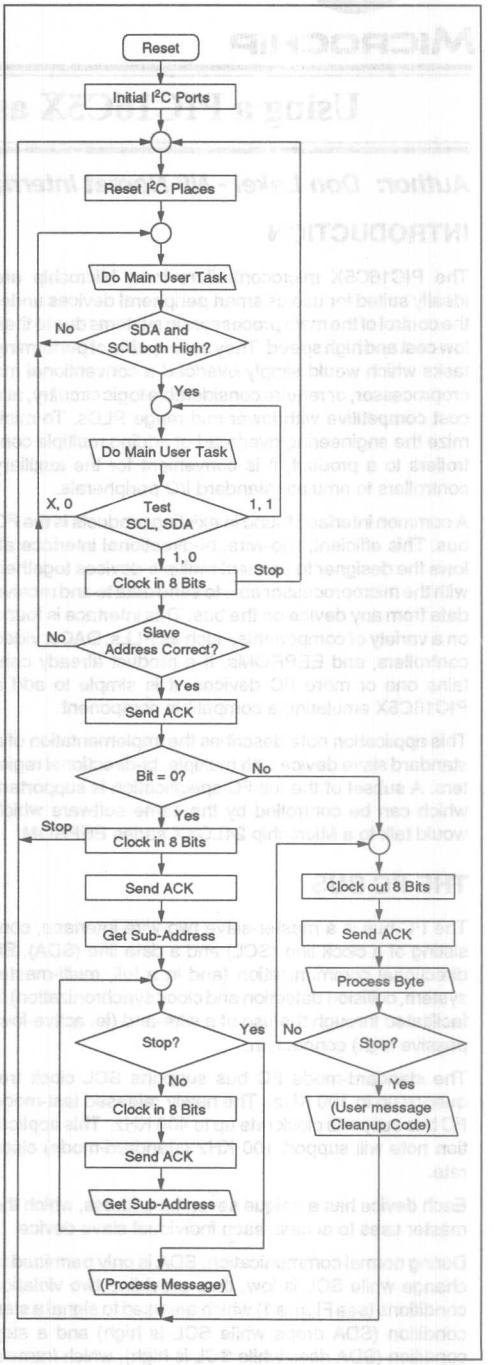
FIGURE 2 - SCHEMATIC OF I²C CONNECTIONS



The connections to the device are shown in Figure 2. The use of RA0 for data in is required. Data is shifted directly out of the port. The code could be modified to make it port independent, but the loss of efficiency may hinder some real-time applications.

This application emulates an I²C device with 8 registers, accessed as sub-addresses 1 through 8 (modulo 7), plus a data channel (0). The example code returns an ID string when the data channel is accessed. When bytes are written to sub-addresses other than 0, they are stored in I2CR0-I2CR7 (I2CR0 gets data written to sub-address 8).

FIGURE 3 - I²C DEVICE FLOWCHART



© 1993 Microchip Technology Inc.

When the initial sub-address is 0, the flag B:ID is set. This is used to indicate access to a special channel. In this case, the data channel is used to return an ID message, or output data to port B, however the natural extension would be to use this as a data I/O channel.

To make the basic device routines easily adaptable to a variety of uses, macros are used to implement the application specific code. This allows the developer the option of using subroutine calls, or in-line code to avoid the 4 clock cycle overhead and use of the precious stack.

Macro	User code function
USER_MAIN	Code to execute in the main loop while not in a message. If this code takes too long, tSH of 4µs will be violated (see Fig. 1). The slave will simply miss the address, not acknowledge, and the master will retry.
USER_Q	This would be quick user code to implement real-time processes. In most applications, this macro would be empty. If used, this routine should be kept under 4µs if possible.
USER_MSG	This would be user code to process a message. It is inserted after a message is successfully received.
USER_RECV	This would be user code to process a received byte. It allows the user to add extra code to implement special purpose sub-addresses such as FIFOs.
USER_XMIT	This would be user code to prepare an output byte. In the default routine, it traps sub-address 0 and calls the ID string function.

References:

I²C Bus Specification, Phillips Components, December 1988.

The I²C bus and how to use it (including specification), Signetics/Philips Semiconductors, January 1992.

Fenger, Carl, "The Inter-Integrated Circuit (I2C) Serial Bus: Theory and Practical Consideration", *Application Note 168*, Philips Components, December 1988.

"24C16 16K CMOS Serial Electrically Erasable PROM", *Microchip Data Book (1992).*

About the Author:

Don Lekei has been designing microprocessor based products over 14 years. He has developed many software and hardware products for a wide variety of applications. Mr. Lekei is Manager of Advanced Technologies at NII Norsat International Inc. at their Canadian headquarters in Surrey, British Columbia. Norsat designs and manufactures products to receive broadcast communications from satellites, terrestrial broadcasting systems and optical fibre. Norsat develops technologies and products for satellite entertainment television, broadcast music and data networks.

© 1993 Microchip Technology Inc.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1

                        LIST    P=16C54, C=80, N=0, R=DEC
0676                    CPU     EQU     1654
0000                    SIM     EQU     0               ;Change timing constants for simulator

                        IF      (CPU==1654) || (CPU==1655)
01FF                    _RESVEC EQU     01FFH           ;16c54 start address
                        ENDIF

                        IF      CPU==1656
                        _RESVEC EQU     03FFH           ;16C56 start address
                        ENDIF

                        IF      CPU==1657
                        _RESVEC EQU     07FFH           ;16C57 start address
                        ENDIF

                        ;*** Reset Vector ****************************************

                        ORG     _RESVEC         ;
                        RESVEC                  ;
01FF 0A0B               GOTO    INIT            ;

                        ;*************************************************************

                        ;*************************************************************

                        ;* Macros to set/clear/branch/skip on bits
                        ;* These macros define and use synthetic "bit labels"
                        ;* Bit labels contain the address and bit of a location
                        ;*
                        ;*************************************************************

                        ;*      Usage                   Description
                        ;*      ----------  ------------

                        ;*      BIT     label,bit,file  ;Define a bit label
                        ;*      SEB     label           ;set bit using bit label
                        ;*      CLB     label           ;clear bit using bit label

                        ;*      SKBS    label           ;SKIP on bit set
                        ;*      SKBC    label           ;SKIP on bit clear
                        ;*      BBS     label,address   ;BRANCH on bit set
                        ;*      BBC     label,address   ;BRANCH on bit clear
                        ;*      CBS     label,address   ;CALL on bit set
                        ;*      CBC     label,address   ;CALL on bit clear
                        ;*
                        ;*************************************************************

                        BIT     MACRO   label,bit,file  ;Define a bit label
                        label   EQU     file<<8|bit
                                ENDM                    ;

                        SEB     MACRO   label           ;Set bit
                                BSF     label>>8,label&7;(macro)
                                ENDM                    ;
```


© 1993 Microchip Technology Inc.

```
                CLB     MACRO   label                   ;Clear bit
                        BCF     label>>8,label&7        ;(macro)
                        ENDM                            ;

                SKBS    MACRO   label                   ;Skip on bit set
                        BTFSS   label>>8,label&7        ;(macro)
                        ENDM

                SKBC    MACRO   label                   ;Skip on bit clear
                        BTFSC   label>>8,label&7        ;(macro)
                        ENDM

                BBS     MACRO   label,address           ;Branch on bit set
                        BTFSC   label>>8,label&7        ;(macro)
                        GOTO    address                 ;(macro)
                        ENDM                            ;

                BBC     MACRO   label,address           ;Branch on bit clear
                        BTFSS   label>>8,label&7        ;(macro)
                        GOTO    address                 ;(macro)
                        ENDM

                CBS     MACRO   label,address           ;Call on bit set
                        CALL    label>>8,label&7        ;(macro)
                        ENDM                            ;

                CBC     MACRO   label,address           ;Call on bit clear
                        CALL    label>>8,label&7        ;(macro)
                        ENDM

                ;For Assembler portability

0000            W       EQU     0                       ;For file,W
0000            w       EQU     0                       ;For file,W
0001            F       EQU     1                       ;For file,F
0001            f       EQU     1                       ;For file,F

                ;*****************************************************

                ;* REGISTER DECLARATIONS
                ;*****************************************************


                        ORG     0                       ;ORG for register declaration

0000  0001      ind     RES     1                       ;0=pseudo-reg 0 for indirect

0001  0001      RTCC    RES     1                       ;1=real time counter
0002  0001      PC      RES     1                       ;2=PC
0003  0001      STATUS  RES     1                       ;3=status reg

                ;* Status reg bits

                        BIT     B_C,0,STATUS            ;Carry
0300            B_C     EQU     STATUS<<8|0


                        BIT     B_DC,1,STATUS           ;Half carry
0301            B_DC    EQU     STATUS<<8|1


                        BIT     B_Z,2,STATUS            ;Zero
0302            B_Z     EQU     STATUS<<8|2
```

© 1993 Microchip Technology Inc.

```
                        BIT     B_PD,3,STATUS   ;Power down
0303                B_PD    EQU     STATUS<<8|3

                        BIT     B_TO,4,STATUS   ;Timeout
0304                B_TO    EQU     STATUS<<8|4

                        BIT     B_PA0,5,STATUS  ;Page select (56/57 only)
0305                B_PA0   EQU     STATUS<<8|5

                        BIT     B_PA1,6,STATUS  ;Page select (56/57 only)
0306                B_PA1   EQU     STATUS<<8|6

                        BIT     B_PA2,7,STATUS  ;GP flag
0307                B_PA2   EQU     STATUS<<8|7


0004  0001          FSR     RES     1               ;4=file select reg 0-4=indirect address
0005  0001          PORTA   RES     1               ;5=port A I/O register (4 bits)
0006  0001          PORTB   RES     1               ;6=port B I/O register

                    IF      (CPU==1655)||(CPU==1657)
                    PORTC   RES     1               ;7=I/O port C on 16C54/56 only
                    ENDIF

                                                    ;registers used by this code

0007  0001          I2CFLG  RES     1               ;I2C flag reg
                                                    ;-i2c flags----------------------

                        BIT     B_RD,0,I2CFLG   ;Flag: 1=read
0700                B_RD    EQU     I2CFLG<<8|0

                        BIT     B_UA,1,I2CFLG   ;Flag: 0=reading unit address
0701                B_UA    EQU     I2CFLG<<8|1

                        BIT     B_SA,2,I2CFLG   ;Flag: 1=reading subabbress
0702                B_SA    EQU     I2CFLG<<8|2

                        BIT     B_ID,3,I2CFLG   ;Flag: 1=reading id
0703                B_ID    EQU     I2CFLG<<8|3

                    ;-------------------------

0008  0001          I2CREG  RES     1               ;I2C I/O register
0009  0001          I2CSUBA RES     1               ;Subaddress
000A  0001          I2CBITS RES     1               ;I2C xmit bit counter

                    ;*********************************************************
                    ;* 8 Pseudo registers accessed by sub-addresses 1-8
                    ;* (address 0 accesses the ID string)
                    ;* these are read-write registers
                    ;*********************************************************


000B                I2CR0   EQU     $               ;Sub-address 8
000B  0001                  RES     1               ;8 pseudo registers

000C                I2CR1   EQU     $               ;Sub-address 1
```


© 1993 Microchip Technology Inc.

```
000C  0001            RES   1

000D                  I2CR2 EQU   $              ;Sub-address 2
000D  0001            RES   1

000E                  I2CR3 EQU   $              ;Sub-address 3
000E  0001            RES   1

000F                  I2CR4 EQU   $              ;Sub-address 4
000F  0001            RES   1

0010                  I2CR5 EQU   $              ;Sub-address 5
0010  0001            RES   1

0011                  I2CR6 EQU   $              ;Sub-address 6
0011  0001            RES   1

0012                  I2CR7 EQU   $              ;Sub-address 7
0012  0001            RES   1

                      ;Constants used by program

00D6                  DEVICE_ADDRESS EQU   0D6H  ;I2C device address


                      ;*************************************************************
                      ;** PORTA DEFINITIONS
                      ;** I2C interface uses PORTA
                      ;** note SDA goes to A0 for code efficiency
                      ;**
                      ;*************************************************************


00F7                  TAREAD  EQU   B'11110111'  ;TRISA register for SDA rea

00F6                  TAWRITE EQU   B'11110110'  ;TRISA register for SDA wri

00F7                  TAINIT  EQU   TAREAD       ;Initial TRISA value

                      BIT   B_SDA,0,PORTA  ;I2C SDA (data) This must be bit 0!
0500                  B_SDA   EQU   PORTA<<8|0


                      BIT   B_SCL,1,PORTA  ;I2C SCL (clock)
0501                  B_SCL   EQU   PORTA<<8|1


                      ;spare        B_???,2,PORTA  ;not used
                      ;spare        B_???,3,PORTA  ;not used

                      ;*************************************************************
                      ;**
                      ;** Port B definition (Parallel out)
                      ;**
                      ;*************************************************************

0000                  TBINIT  EQU   B'00000000'  ;Port B tris (all output)
00FF                  PBINIT  EQU   B'11111111'  ;Port B init


                      ;*************************************************************
                      ;* Macros to contain user POLL loop code.
                      ;* These are implimented as macros to allow ease of modification,
                      ;* especially in real-time applications. The functions could be coded as
                      ;* in-line code or as subroutines depending on ROM/time tradeoffs.
                      ;*
                      ;* USER_MAIN:   Decision or code to perform at idle time
                      ;*
```

© 1993 Microchip Technology Inc.

```
                ;* USER_Q:       'Quick' code for use during transfer - max
                ;*              I²C Spec. More than 4 µs may result in I²C
                ;*              full spec speed.
                ;*
                ;* USER_MSG:     Code to execute at receipt of I²C command.
                ;*
                ;************************************************************

                USER_MAIN       MACRO
                ;*** This would be user code for idle loop
                                ENDM

                USER_Q          MACRO
                ;*** This would be quick user code
                                ENDM

                USER_MSG        MACRO
                ;*** This would be user code to process a message
                                ENDM

                USER_RECV       MACRO
                ;*** This would be user code to process a received byte
                ;*** example code sends sub-address 0 to port b
                                BBC     B_ID,_NXI_notid             ;Channel 0! Bit set if

                                MOVFW   I2CREG                      ;get received byte
                                MOVWF   PORTB                       ;and write it on portb
                                GOTO    IN_CONT
                _NXI_notid
                                ENDM

                USER_XMIT       MACRO
                ;*** This would be user code to prepare an output byte
                ;*** example code sends id string to output
                                BBC     B_ID,_NXO_notid             ;Channel 0! Bit set if

                                CALL    GETID                       ;get next byte from ID
                                GOTO    OUT_CONT                    ;and send it
                _NXO_notid
                                ENDM
                ;************************************************************
                ; START OF CODE
                ;************************************************************
                        ORG     0
                ;************************************************************

                ;* Device ID Table (must be at start)
                ;* TABLE FOR UNIT ID returns next char in W
                ;************************************************************

                GETID
0000 0209               MOVFW   I2CSUBA                             ;W=I2CSUBA
0001 0E07               ANDLW   07H                                 ;Limit to 8 locations
0002 01E2               ADDWF   PC,F
                ;************************************************************
                ;* Device ID text: read starting at sub-address 0
                ;************************************************************


0003 0850               RETLW   'P'
0004 0849               RETLW   'I'
0005 0843               RETLW   'C'
0006 0849               RETLW   'I'
0007 0832               RETLW   '2'
0008 0843               RETLW   'C'
0009 0800               RETLW   0
```


© 1993 Microchip Technology Inc.

```
000A 0800                RETLW   0

                         ;*****************************************************
                         ;* I2C Device routines
                         ;*
                         ;* Enable must be HIGH, else state goes to 0
                         ;* write is to me, read is from me.
                         ;*
                         ;*              <=============== first byte / subsequant write
                         ;*
                         ;* SDA   -|   X--X--X--X--X--X--X--X-
                         ;*          |-X--X--X--X--X--X--X--X-
                         ;* (bit)    s    7     6     5     4     3     2     1     0
                         ;* SCL  --|  |-|   |-|   |-|   |-|   |-|   |-|   |-|   |
                         ;*           |-|   |-|   |-|   |-|   |-|   |-|   |-|   |-|
                         ;*
                         ;* STATE: 0 1 2  3  2  3  2  3  2  3  2  3  2  3  2  3  2
                         ;*
                         ;*              <=============== subsequant reads ============
                         ;*
                         ;* SDA     X--X--X--X--X--X--X--X--X
                         ;*        -X--X--X--X--X--X--X--X--X--X
                         ;* (bit)ack   7     6     5     4     3     2     1     0
                         ;* SCL  -|  |-|   |-|   |-|   |-|   |-|   |-|   |-|   |-|
                         ;*         |-|   |-|   |-|   |-|   |-|   |-|   |-|   |-|   |
                         ;*
                         ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8
                         ;*
                         ;*              <=============== Final READ ==================
                         ;*
                         ;* SDA    X--X--X--X--X--X--X--X--X--X
                         ;*       -X--X--X--X--X--X--X--X--X--X
                         ;* (bit)ack   7     6     5     4     3     2     1     0
                         ;* SCL  -|   |-|   |-|   |-|   |-|   |-|   |-|   |-|   |-|
                         ;*          |-|   |-|   |-|   |-|   |-|   |-|   |-|   |-|   |
                         ;*
                         ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8
                         ;*
                         ;* STATE B is an ignore bit state for non-addressed bits
                         ;* STATE C indicates last sample had ENA low
                         ;* on rising edge of ENA, DATA LOW = low voltage, DATA&CLOC
                         ;*****************************************************
                         ;I2C interface uses PORTA
                         ;note SDA must be on PORTA,0 for code efficiency
                         ;*****************************************************
                         ;** INIT
                         ;** Hardware reset entry point
                         ;**
                         ;*****************************************************
                         INIT    ;Power-on entry
                         ;*****************************************************
                         ;** RESET
                         ;** software reset entry point
                         ;**
                         ;*****************************************************
                         RESET                                   ;Soft reset

000B 0CF7                        OVLW   TAINIT                  ;Init ports
000C 0005                        TRIS    PORTA
000D 0C00                        MOVLW  TBINIT
000E 0006                        TRIS    PORTB
000F 0CFF                        MOVLW   PBINIT
0010 0026                        MOVWF   PORTB
                         ;*****************************************************
                         ; Main wait loop while idle. POLL loop should be called here
                         ;
```

```
                        ;*****************************************************
                        I2CWAIT
0011 0004                       CLRWDT                          ;Clear watchdog timer
                                CLB     B_UA                    ;Init state flags
0012 0427                       BCF     B_UA>>8,B_UA&7
                                CLB     B_SA                    ;Init state flags
0013 0447                       BCF     B_SA>>8,B_SA&7
                                CLB     B_RD                    ;Init state flags
0014 0407                       BCF     B_RD>>8,B_RD&7
                        loop1
0015 0004                       CLRWDT                          ;Clear watchdog timer
                        USER_MAIN                               ;Call user code while in idle state
                        ;*** This would be user code for idle loop
                                SKBC    B_SDA                   ;Wait for SDA&SCL=H
0016 0605                       BTFSC   B_SDA>>8,B_SDA&7
                        loop2
                                SKBS    B_SCL                   ;
0017 0725                       BTFSS   B_SCL>>8,B_SCL&7
0018 0A15                       GOTO    loop1                   ; No longer valid to wait f
0019 0004                       CLRWDT                          ;Clear watchdog timer
                                USER_MAIN                       ;Call user code while in idle state
                        ;*** This would be user code for idle loop
                        ;** wait for start **
                                SKBC    B_SCL                   ;Clock has dropped
001A 0625                       BTFSC   B_SCL>>8,B_SCL&7
                                SKBC    B_SDA                   ;Data dropped... Start!
001B 0605                       BTFSC   B_SDA>>8,B_SDA&7
001C 0A17                       GOTO    loop2
                        ;** START RECEIVED! - wait for first bit!
                        loop3
                                BBS     B_SDA,I2CWAIT           ;Data raised before clock dropped -
001D 0605                       BTFSC   B_SDA>>8,B_SDA&7
001E 0A11                       GOTO    I2CWAIT
                                BBS     B_SCL,loop3             ;Wait for clock low
001F 0625                       BTFSC   B_SCL>>8,B_SCL&7
0020 0A1D                       GOTO    loop3
                        NEXTBYTE
0021 0004                       CLRWDT                          ;Clear watchdog timer
0022 0C01                       MOVLW   1                       ;Init receive byte so bit f
0023 0028                       MOVWF   I2CREG
                        ;** Shift bits! - external poll may be executed during low
                        ;* ENABLE line is checked for loss of enable ONLY during HI
                        ;*** CLOCK IS LOW - DATA MAY CHANGE HERE
                        ;*** We have at least 4 Ês before any change can occur
                        loop4
                                USER_Q
                        ;*** This would be quick user code
                        loop4A
                                BBC     B_SCL,loop4A            ;Wait for clock high
0024 0725                       BTFSS   B_SCL>>8,B_SCL&7
0025 0A24                       GOTO    loop4A
                        ;*** CLOCK IS HIGH - SHIFT BIT - then watch for change
0026 0305                       RRF     PORTA,W                 ;Move RA0 into C
0027 0368                       RLF     I2CREG,F                ;Shift in bit
0028 0603                       SKPNC                           ;Skip if not done
0029 0A36                       GOTO    ACK_I2C                 ;Acknowledge byte
002A 0608                       BTFSC   I2CREG,0                ;Skip if data bit was 0
002B 0A31                       GOTO    ii_1                    ;This bit was set
                        ii_0
                                BBC     B_SCL,loop4             ;Wait for clock  low
002C 0725                       BTFSS   B_SCL>>8,B_SCL&7
002D 0A24                       GOTO    loop4
                                SKBS    B_SDA                   ;Data low-high == stop
002E 0705                       BTFSS   B_SDA>>8,B_SDA&7
002F 0A2C                       GOTO    ii_0
                                I2CSTOP
                                USER_MSG                        ;process completed message!
                        ;*** This would be user code to process a message
0030 0A11                       GOTO    I2CWAIT                 ;back to main loop
```


© 1993 Microchip Technology Inc.

```
                ii_1    BBC     B_SDA,I2CWAIT           ;Data high-low == start
0031 0705               BTFSS   B_SDA>>8,B_SDA&7
0032 0A11               GOTO    I2CWAIT
                        BBC     B_SCL,loop4             ;Wait for clock  low
0033 0725               BTFSS   B_SCL>>8,B_SCL&7
0034 0A24               GOTO    loop4
0035 0A31               GOTO    ii_1
              ACK_I2C
                        BBC     B_UA,ACK_UA             ;Not addressed - check unit address
0036 0727               BTFSS   B_UA>>8,B_UA&7
0037 0A8B               GOTO    ACK_UA
                        BBS     B_SA,ACK_SA             ;Reading secondary address
0038 0647               BTFSC   B_SA>>8,B_SA&7
0039 0A97               GOTO    ACK_SA
              ;****
              ;** Do what must be done with new data bytes here (before A

              ;** Don't ack if byte can't be processed!
              ;****
              ;----
              USER_RECV
              ;*** This would be user code to process a received byte
              ;*** example code sends sub-address 0 to port b
003A 0767               BTFSS   B_ID>>8,B_ID&7
003B 0A3F               GOTO    _NXI_notid
003C 0208               MOVFW   I2CREG                  ;get received byte
003D 0026               MOVWF   PORTB                   ;and write it on portb
003E 0A47               GOTO    IN_CONT
              _NXI_notid
003F 0C07               MOVLW   07H                     ;Register count
0040 0169               ANDWF   I2CSUBA,f               ;Limit register count
0041 0C0B               MOVLW   I2CR0                   ;Pseudo-registers
0042 01C9               ADDWF   I2CSUBA,W               ;Offset from buffer start
0043 02A9               INCF    I2CSUBA                 ;Next sub-address
0044 0024               MOVWF   FSR                     ;Indirect address
0045 0208               MOVFW   I2CREG
0046 0020               MOVWF   ind                     ;Put data into register
              IN_CONT                                   ;continue point for interce
              ACKloop
                        BBS     B_SCL,ACKloop           ;Wait for clock low
0047 0625               BTFSC   B_SCL>>8,B_SCL&7
0048 0A47               GOTO    ACKloop
                        CLB     B_SDA                   ;Set ACK
0049 0405               BCF     B_SDA>>8,B_SDA&7
004A 0CF6               MOVLW   TAWRITE
004B 0005               TRIS    PORTA
                        CLB     B_SDA                   ;Set ACK (just in case docs are wrong)
004C 0405               BCF     B_SDA>>8,B_SDA&7
              ACKloop2
                       USER_Q
              ;*** This would be quick user code
                        BBC     B_SCL,ACKloop2          ;Wait for clock high
004D 0725               BTFSS   B_SCL>>8,B_SCL&7
004E 0A4D               GOTO    ACKloop2
              ACKloop3
                      USER_Q
              ;*** This would be quick user code
                        BBS     B_SCL,ACKloop3          ;Wait for clock low
004F 0625               BTFSC   B_SCL>>8,B_SCL&7
0050 0A4F               GOTO    ACKloop3
0051 0CF7               MOVLW   TAREAD                  ;End ACK
0052 0005               TRIS    PORTA
                        BBC     B_RD,NEXTBYTE           ;Skip if read (we were acking address or
0053 0707               BTFSS   B_RD>>8,B_RD&7
0054 0A21               GOTO    NEXTBYTE
              ;*********************************************************
              ; I2C Readback (I2C read request)
              ; Application specific code to get bytes to send may be add
```

```
                        ; This routine gets data from location pointed to by I2CSUB
                        ; sends it to I2C. Subsequent reads get sequential addresse
                        ; AND's the register # with 7 to limit to 8 registers (for
                        ; could be modified to do a comparison to an ablolute numbe
                        ;
                        ;*********************************************************
                        NEXTOUT
                        ;*** <<< PUT NEXT BYTE INTO I2CREG HERE NOW! >>> ***
                        USER_XMIT
                        ;*** This would be user code to prepare an output byte
                        ;*** example code sends id string to output
0055 0767                       BTFSS   B_ID>>8,B_ID&7
0056 0A59                       GOTO    _NXO_notid
0057 0900                       CALL    GETID                   ;get next byte from ID chan
0058 0A60                       GOTO    OUT_CONT                ;and send it
                        _NXO_notid
0059 0C07                       MOVLW   07H                     ;Register count
005A 0169                       ANDWF   I2CSUBA,f               ;Limit register count
005B 0C0B                       MOVLW   I2CR0                   ;Pseudo-registers
005C 01C9                       ADDWF   I2CSUBA,W               ;Offset from buffer start
005D 02A9                       INCF    I2CSUBA                 ;Next sub-address
005E 0024                       MOVWF   FSR                     ;Indirect address
005F 0200                       MOVFW   ind                     ;Get data from register
                        OUT_CONT
0060 0028                       MOVWF   I2CREG
                        ;- add code here to init I2CREG! when B_ID is clear!
0061 0C08                       MOVLW   8                       ;Bit counter
0062 002A                       MOVWF   I2CBITS
                        ;** OUT bits! - external poll may be executed during low c
                        ;                may also be executed during high cycle if
                        ;* ENABLE line is checked for loss of enable ONLY during HI
                        ;*** CLOCK IS LOW - CHANGE DATA HERE FIRST!
                        ;*** loop 1: data was 1
                        iiOUT_loop_1
0063 0368                       RLF     I2CREG,F                ;Shift data out, MSB first
0064 0603                        SKPNC                          ;1->0: change
0065 0A79                        GOTO    iiOUT_1                ;Output another 1!
                                 CLB     B_SDA                  ;Output 0
0066 0405                        BCF     B_SDA>>8,B_SDA&7
0067 0CF6                        MOVLW   TAWRITE
0068 0005                        TRIS    PORTA
                                 CLB     B_SDA                  ;Set data (just in case docs are
0069 0405                        BCF     B_SDA>>8,B_SDA&7
                        iiOUT_0
006A 0004                       CLRWDT                          ;Clear watchdog timer
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_02
                                BBC     B_SCL,iiOUT_loop_02     ;Wait for clock high
006B 0725                       BTFSS   B_SCL>>8,B_SCL&7
006C 0A6B                       GOTO    iiOUT_loop_02
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_03
                                BBS     B_SCL,iiOUT_loop_03     ;Wait for clock low
006D 0625                       BTFSC   B_SCL>>8,B_SCL&7
006E 0A6D                       GOTO    iiOUT_loop_03
006F 02EA                       DECFSZ  I2CBITS                 ;Count bits
0070 0A74                       GOTO    iiOUT_loop_0            ;Loop for last bit 0
0071 0CF7                       MOVLW   TAREAD                  ;Done with last bit 0... Se
0072 0005                       TRIS    PORTA
0073 0A80                       GOTO    iiOUT_ack               ;Get ACK
                        iiOUT_loop_0
0074 0368                       RLF     I2CREG,F                ;Shift data out, MSB first
0075 0703                       SKPC                            ;0->1: change
0076 0A6A                       GOTO    iiOUT_0                 ;Output another 0!

0077 0CF7                       MOVLW   TAREAD                  ;Set to 1
0078 0005                       TRIS    PORTA
```


© 1993 Microchip Technology Inc.

```
                        iiOUT_1
0079 0004                       CLRWDT                          ;Clear watchdog timer
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_12
                                BBC     B_SCL,iiOUT_loop_12     ;Wait for clock high
007A 0725                               BTFSS  B_SCL>>8,B_SCL&7
007B 0A7A                       GOTO    iiOUT_loop_12
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_13
                                BBS     B_SCL,iiOUT_loop_13     ;Wait for clock low
007C 0625                       BTFSC   B_SCL>>8,B_SCL&7
007D 0A7C                       GOTO    iiOUT_loop_13
007E 02EA                       DECFSZ  I2CBITS                 ;Count bits
007F 0A63                       GOTO    iiOUT_loop_1            ;Loop for last bit 1
                        iiOUT_ack                               ;Get acknowledge
0080 02A9                       INCF    I2CSUBA                 ;Next sub-address
                        iiOUT_loop_a2
                                BBC     B_SCL,iiOUT_loop_a2     ;Wait for clock high
0081 0725                       BTFSS   B_SCL>>8,B_SCL&7
0082 0A81                       GOTO    iiOUT_loop_a2
                                BBS     B_SDA,I2CWAIT           ;No ACK - wait for restart!
0083 0605                       BTFSC   B_SDA>>8,B_SDA&7
0084 0A11                       GOTO    I2CWAIT
                        ;- prepare next character here!
                        iiOUT_loop_a3
                                BBC     B_SCL,NEXTOUT           ;Wait for clock low - output next!
0085 0725                       BTFSS   B_SCL>>8,B_SCL&7
0086 0A55                       GOTO    NEXTOUT
                                BBS     B_SDA,iiOUT_loop_a3     ;Watch out for new start condition!
0087 0605                       BTFSC   B_SDA>>8,B_SDA&7
0088 0A85                       GOTO    iiOUT_loop_a3
0089 0A11                       GOTO    I2CWAIT                 ;Stop received!
008A 0A11                       GOTO    I2CWAIT
                        ;*********************************************************
                        ;* Unit address received - check for valid address
                        ;*
                        ;*********************************************************
                        ACK_UA
                                 SEB     B_UA                   ;Flag unit address received
008B 0527                        BSF     B_UA>>8,B_UA&7
008C 0608                        BTFSC   I2CREG,0               ;Skip if data coming in
                                 SEB     B_RD                   ;Flag - reading from slave
008D 0507                        BSF     B_RD>>8,B_RD&7
008E 0208                        MOVF    I2CREG,W               ;Get address
008F 0EFE                        ANDLW   0FEH                   ;Mask direction flage before compare
0090 0FD6                        XORLW   DEVICE_ADDRESS         ;Device address
0091 0743                        BNZ     I2CWAIT                ;Not for me! (skip rest of message)
0092 0A11                        BBS     B_RD,ACKloop           ;Read - no secondary address
0093 0607                        BTFSC   B_RD>>8,B_RD&7
0094 0A47                        GOTO    ACKloop
                                 SEB     B_SA                   ;Next is secondary address
0095 0547                        BSF     B_SA>>8,B_SA&7
0096 0A47                        GOTO     ACKloop               ;Yes! ACK address and continue
                        ;*********************************************************
                        ;* Secondary address received - stow it!
                        ;* SA = 0 is converted to 128 to facilitate ID read
                        ;*********************************************************
                                ACK_SA
                                 CLB     B_SA                   ;Flag second address received
0097 0447                        BCF     B_SA>>8,B_SA&7
                                 CLB     B_ID
0098 0467                        BCF     B_ID>>8,B_ID&7
0099 0208                        MOVFW    I2CREG                ;Get subaddress
009A 0643                        SKPNZ                          ;Not 0
                                 SEB     B_ID                   ;Flag - id area selected
009B 0567                        BSF     B_ID>>8,B_ID&7
```

© 1993 Microchip Technology Inc.

```
009C 0029                       MOVWF   I2CSUBA             ;Set subaddress
009D 0A47                       GOTO    ACKloop
                                END


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

PIC16C54A EMI Results

INTRODUCTION

This paper discusses the EMI results of the PIC16C54A. These measurements were taken by an independent consulting firm that specializes in electromagnetic testing. These results are for a specific system design, each design will have its own results.

DEVICES USED

These tests were done on a random PIC16C54A device, and should be considered as typical results. Initial testing was done on three boards / devices. The device frequency of the boards were, respectively, 32 KHz, 4 MHz, and 20 MHz. As would be expected there was a substantial difference (decrease) at the low frequency as compared to the higher frequencies. The difference between the 4 MHz operation and the 20 MHz operation was marginal. The final testing of the device was done at 4 MHz, and was according to the FCC measurement procedure MP-4.

SYSTEM USED

The PIC16C54A device was tested in the Microchip OHMMETER board. This board is a three layer board, with a ground plane. Power and ground planes greatly help in the compliance of designs to the FCC part 15 subpart B testing. This board had minimal other external components, so that the electromagnetic measurements could mostly be attributed to the device and design of the system. To reduce the noise that comes from a power supply, a 10 nF bypass capacitor was attached to the power / ground of the input jack.

EMI TESTING

The testing of electromagnetic noise (EM) on a system has two types of commonly used testing environments, an internal and external environment. The internal test is done is a screen room to reduce the amount of stray EMI. The indoor tests is useful in determining the source of EMI radiation. The external test is done outdoors. This places the equipment under test in an environment to measure radiated emissions (EMI). The FCC only requires the outdoor testing of devices. The equipment under test (EUT) was positioned to maximize the emissions.

INTERNAL TESTS

The equipment under test was performed on a wooden test bench, inside a screen room, 0.8m above the earth ground plane (see Figure 1). The EUT was powered through the Line Impedance Stabilization Network (LISN) bonded to the ground plane. The LISN power was filtered and the filter was bonded to the ground plane. The EUT was positioned on the table with the minimum distance from any conductive surface, as specified in MP-4. The excess power cord was wrapped in a figure-8 pattern to form a bundle approximately 8 cm in length. The EUT configuration was set for the highest emission frequency, and data was collected under the program control of the host computer. The spectrum analyzer collected the maximum peak readings over each spectrum. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 1.

TABLE 1: CONDUCTED EMISSIONS RESULTS

Frequency (MHz)	Emission Level dBuV	Emission Level uV	Specification Limit uV
0.5345	33.2	46	250
7.085	33.1	45	250
10.08	33.1	45	250
10.42	33.3	46	250
11.62	33.4	47	250
13.41	33.8	49	250

© 1994 Microchip Technology Inc.

While in the screen room additional analyses on the device was done. First a local probe was used to test the emission levels around the board and device. There were no measurable emissions at the I/O pins or their corresponding trace lines. Emissions were measured at the jack to the power supply. The power supply and cord were the greatest source of emissions for the EUT. This is due to the power cord being an antenna which emitted the noise from the power supply. Designers should attempt to minimize antennas, which emit EMI. Antennas could be the power supply cord, as well as traces on the system board.

Second the PIC16C54A was monitored for susceptibility, following the IEC 801-3 specification. This test was measured from 27 MHz to 500 MHz in 10 KHz steps. The device did not display any signs of susceptibility (see Table 2).

EXTERNAL TESTS

The open field site used for radiated emission testing was setup according to the FCC bulletin OST 55. Figure 2 shows the layout of the open field test site. The EUT was mounted on a turntable. The position of the turntable is remote controlled to determine the highest emission levels. Initial testing was done with a broad band mounted on the antenna mast distance of 3 meters. Further investigation was done to determine the EUT positions that produced the maximum level of emissions. The receiving antenna was mounted on the antenna mast. The antenna height was varied to find the highest level of radiated emissions at each frequency. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 3. Figures 3 and 4 show the dBuV vs. MHz graphs.

TABLE 2: RADIATED SUSCEPTIBILITY

Frequency (MHz)	SPEC V/M	Threshold V/M	Modulation	Comments
27.0 - 500.0 †	3.0	> 3.0	80%	No Susceptibility

† Frequency incremmented in 10 KHz steps

TABLE 3: RADIATED EMISSIONS RESULTS

Frequency	Meter Reading (dBuV)	Antenna Factor dB	Effective Gain dB	Distribution Factor dB	Corr. Rdg dBuV/m	Corr. Rdg. uV/m	Spec Limit uV/m
36.01	53.1	11.8	34.0	0	30.9	35	100
40.04	53.0	11.3	34.2	0	30.1	32	100
44.04	56.1	11.1	34.0	0	33.2	46	100
48.04	55.1	11.0	33.7	0	32.4	42	100
64.05	55.2	9.0	33.9	0	30.3	33	100
112.05	56.0	10.6	33.4	0	33.2	46	150

© 1994 Microchip Technology Inc.

FIGURE 1

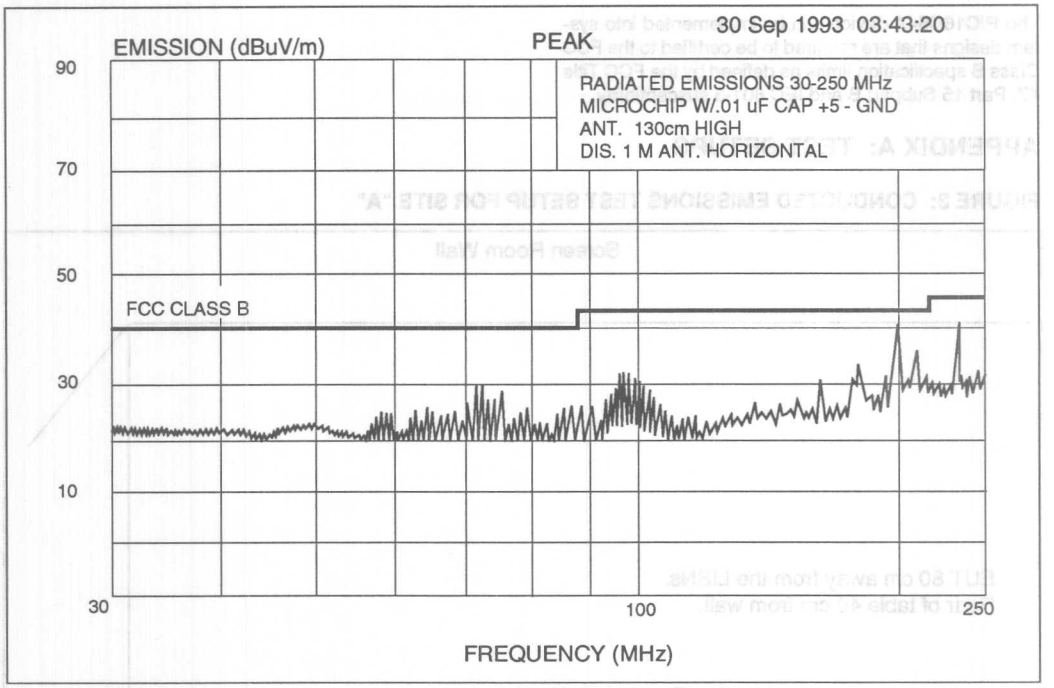
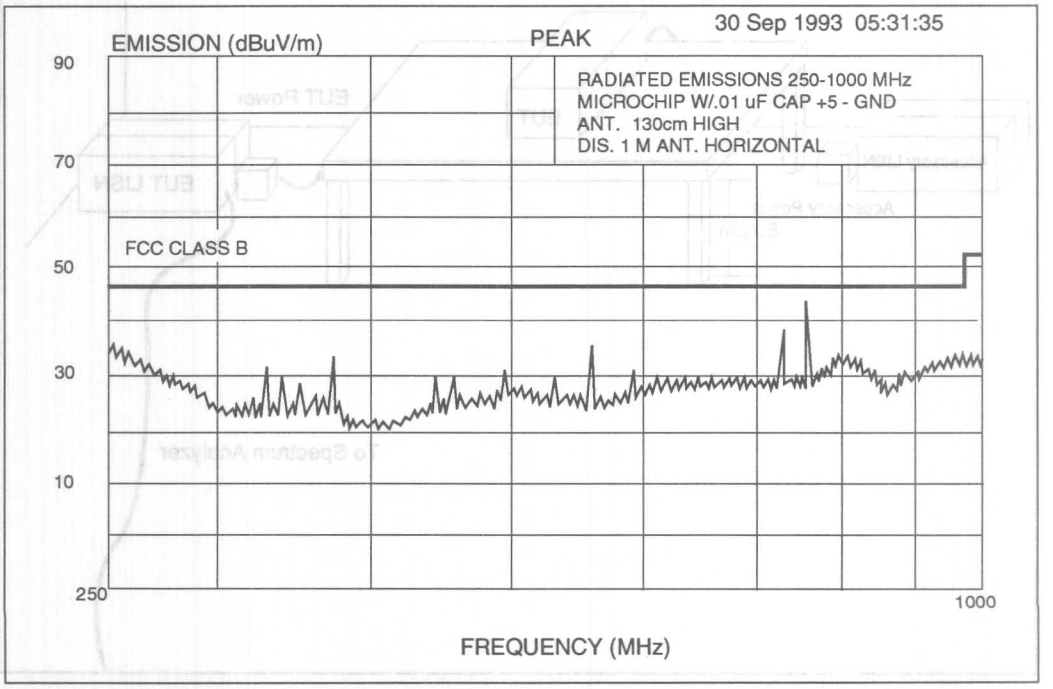


FIGURE 2



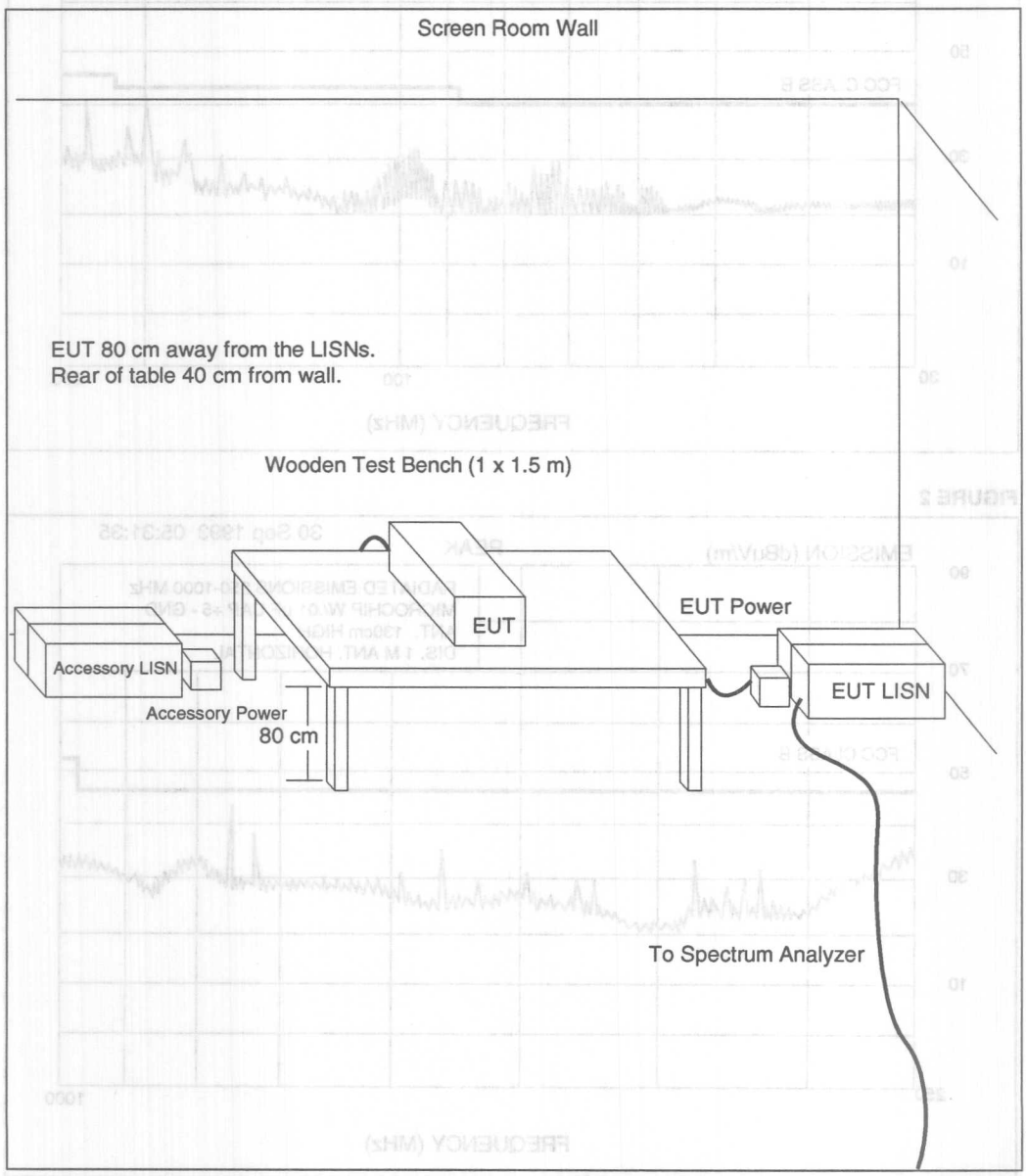
© 1994 Microchip Technology Inc.

CONCLUSION

The PIC16C54A device can be implemented into system designs that are required to be certified to the FCC Class B specification limits as defined by the FCC Title 47, Part 15 Subpart B and IEC 801-3 susceptibility.

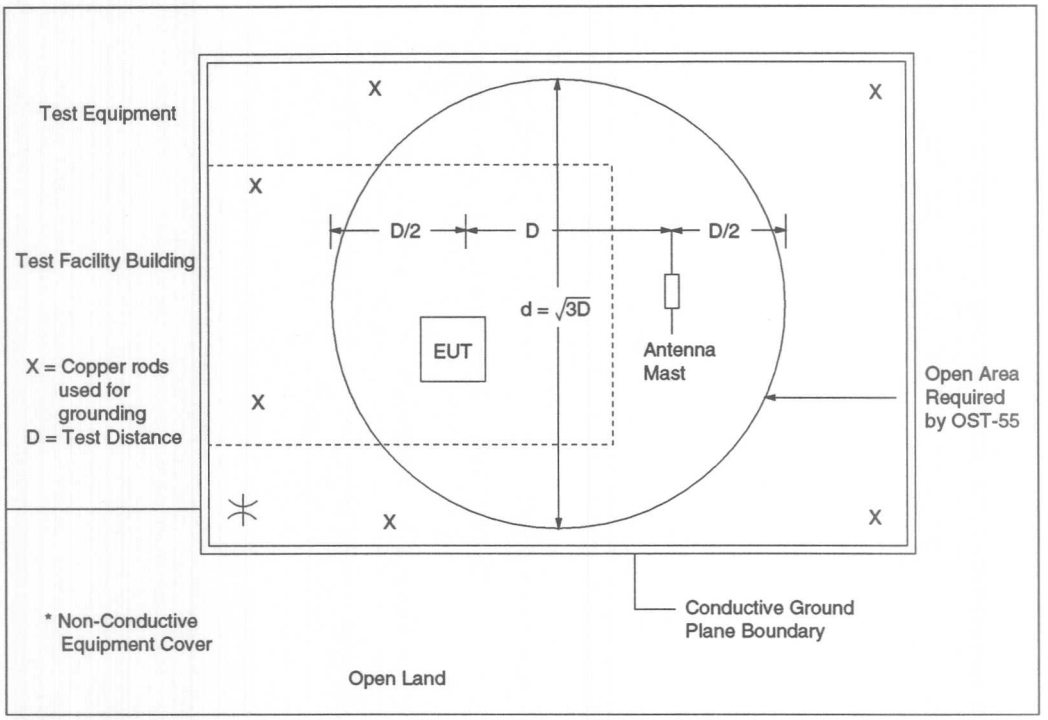
APPENDIX A: TEST SETUPS

FIGURE 3: CONDUCTED EMISSIONS TEST SETUP FOR SITE "A"



© 1994 Microchip Technology Inc.

FIGURE 4: RADIATED EMISSIONS TEST SETUP FOR SITE "A"



OHMMETER Board Setup:	*Stan D'Souza*
Written By:	*Mark Palmer*
Testing By:	*Compatible Electronics*

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

SECTION 3
PIC16CXX APPLICATION NOTES

Using the PortB Interrupt on Change as an External Interrupt - AN566 ..... 3- 1
Implementing Wake Up on Keystroke - AN552 ..... 3- 5
Using the 8-Bit Parallel Slave Port - AN579 ..... 3- 9
A PC-Based Development Programmer for the PIC16C84 - AN589 ..... 3- 15
Using the CCP Modules - AN594 ..... 3- 21
Interfacing to an LCD Module - AN587 ..... 3- 49
Using Timer1 in Asynchronous Clock Mode - AN580 ..... 3- 95
Low-Power Real Time Clock - AN582 ..... 3- 99
Using the Analog to Digital Converter - AN546 3-123
Four Channel Digital Voltmeter with Display and Keyboard - AN557 3-141
Apple® Desktop Bus - AN591 3-169
Software Implementation of Asynchronous Serial I/O - AN555 3-181
Software Implementation of I²C™ Bus Master - AN554 3-223
Use of the SSP Module in the I²C Multi-Master Environment - AN578 3-297

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

MICROCHIP

AN566

Using the PortB Interrupt on Change as an External Interrupt

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. To address these applications, there is the PIC16CXX microcontroller family of products. This family has numerous peripheral and special features to better address user applications.

One feature is the interrupt on change of the PORTB pins. This "interrupt on change" is caused when any of the RB<7:4> pin, configured as input, changes levels. When used in conjunction with the software programmable weak internal pull-ups, a direct interface to a keypad is possible. This is shown in application note AN552 (Implementing Wake-up on Key Stroke). Another way to use the "interrupt on change" feature would be as additional external interrupt sources. This allows the PIC16CXX devices to support multiple external interrupts, in addition to the INT pin.

This application note will discuss some of the issues in using PortB as additional external interrupt pins, and will show some examples. These examples can be easily modified to suit your particular needs.

USING A PORTB INPUT FOR AN EXTERNAL INTERRUPT

The interrupt source(s) cannot simply be directly connected to the PortB pins, and expect the interrupt to function the same as the interrupt (INT) pin. The characteristic of the interrupt signal must also be known to develop the microcontrollers hardware/software. After we know this, we can determine the best way to structure the program to handle this signal. These characteristics include:

1. Trigger interrupt on rising, falling, or both edges
2. What is the pulse width of the interrupt (high time / low time)

It is easy to understand the need of knowing which edge triggers the external interrupt service routine. This allows one to ensure that the interrupt service routine is only entered for the desired edge, with all other edges ignored. Not so clear is pulse width of the interrupt. This determines the amount of additional overhead that the software routine may need.

3

© 1993 Microchip Technology Inc.

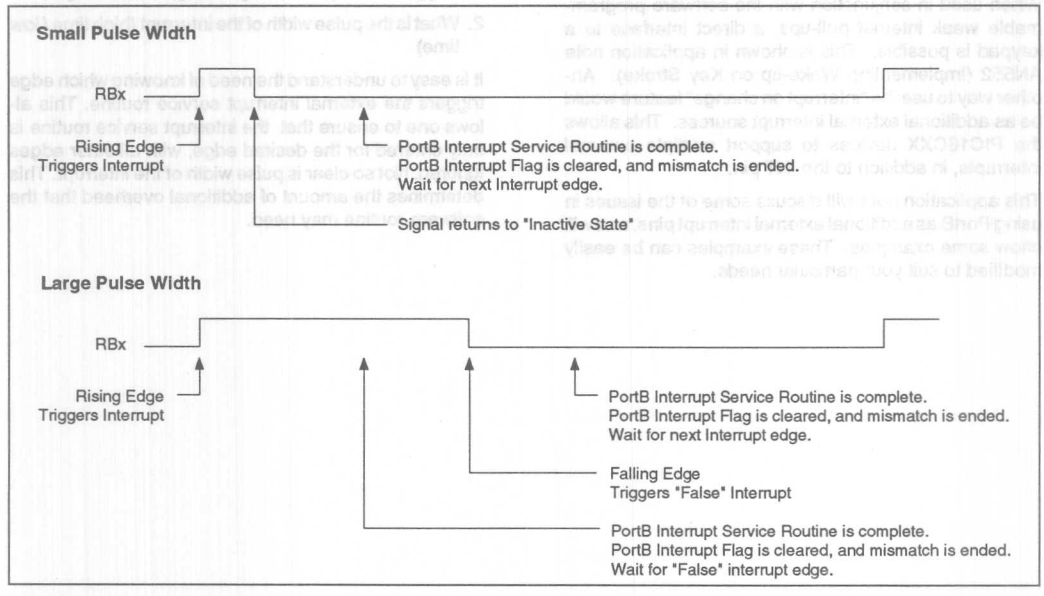
Using the PortB Interrupt on Change as an External Interrupt

Figure 1 shows the two cases for the interrupt signal verses the time to complete the interrupt service routine. The first waveform is when the signal makes the low-to-high-to-low transitions before the interrupt service routine has completed (interrupt flag cleared). When the interrupt flag has been cleared the interrupt signal has already returned to the inactive level. The next transition of the signal is due to another interrupt request. An interrupt signal with this characteristic will be called a small pulse width signal. The second waveform is when the signal only makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). The next transition (high-to-low) will return the interrupt signal to the inactive level. This will generate a "false" interrupt, that will need to be cleared. Then the following transition (low-to-high) will be a "true" interrupt. An interrupt signal with this characteristic will be called a wide pulse width signal.

An interrupt pulse with a small pulse width requires less overhead than a wide pulse width. A small pulse width signal must be less then the minimum execution time of the interrupt service routine, while a wide pulse width must be greater then the maximum time through the interrupt service routine.

Example 1 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a small pulse width. In this case since the interrupt pulse width is small, the pulse has gone high and then low again before PortB is read to end the mismatch condition. So when PortB is read it will read a low signal and will again be waiting for the rising edge transition.

FIGURE 1 - INTERRUPT STEPS FOR SMALL AND WIDE PULSE WIDTHS



Example 1: Single Interrupt with a Small Pulse Width

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
                :                           ; Do task for INT on RB7
                :                           ;
CLR_RBINTF      MOVF    PORTB, 1            ; Read PortB (to itself) to end
                                            ; mismatch condition
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                RETFIE                      ; Return from interrupt
OTHER_INT               :                   ; Do what you need to here
                        :
                RETFIE                      ; Return from interrupt
```


© 1993 Microchip Technology Inc.

Example 2 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a wide pulse width. In this case since the interrupt pulse width is large, the pulse is still high before PortB is read to end the mismatch condition. So when PortB is read it will read a high signal and will generate an interrupt on the next falling edge transition (which should be ignored).

Example 3 shows a interrupt on change with the interrupt source on PortB (RB7). This executes the interrupt service routine on a both edges. The interrupt source must have a minimum pulse width to ensure that both edges can be "seen". The minimum pulse width is the maximum time from the interrupt edge to the reading of PortB and clearing the interrupt flag.

Example 2: Single Interrupt with a Wide Pulse Width

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
                BTFSC   PORTB, RB7          ; Check for rising edge
                GOTO    CLR_RBINTF          ; Falling edge, clear PortB int
                :                           ;   flag
                        :                   ; Do task for INT on RB7
                        :
CLR_RBINTF      MOVF    PORTB, 1            ; Read PortB (to itself) to end
                                            ; mismatch condition
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                RETFIE                      ; Return from interrupt
OTHER_INT               :                   ; Do what you need to here
                        :
                RETFIE                      ; Return from interrupt
```

Example 3: Interrupt on Change

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
CLR_RBINTF      MOVF    PORTB, 1            ; Read PortB (to itself) to end
                                            ; mismatch condition
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                        :                   ; Do task for INT on RB7
                        :                   ;
                RETFIE                      ; Return from interrupt
OTHER_INT               :                   ; Do what you need to here
                        :
                RETFIE                      ; Return from interrupt
```

© 1993 Microchip Technology Inc.

Using PortB Inputs for Multiple Interrupts

The previous examples have been for a single external interrupt on PORTB. This can be extended to support up to 4 external interrupts. To do this requires additional software overhead, to determine which of the PortB pins (RB<7:4>) caused the interrupt. Care should be taken in the software to ensure that no interrupts are lost.

In this example, the interrupt sources on RB7, RB5, and RB4 have a small pulse width, while the interrupt source on pin RB6 is wide and should cause a trigger on the rising edge.

SUMMARY

The PortB interrupt on change feature is both a very convenient method for direct interfacing to an external keypad, with no additional components, but is also versatile in its uses. The ability to add up to four additional external interrupt. Of course hybrid solutions are also possible. That is, for example, using PORTB<6:1> as a 3x3 keypad, with PORTB7 as an external interrupt and PORTB0 as a general purpose I/O. The flexibility of this feature allows the user to implement a best fit design for the application.

Example 4: Multiple Interrupts with Different Pulse Widths

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
;
; PortB change interrupt has occurred. Must determine which pin caused
; interrupt and do appropriate action. That is service the interrupt,
; or clear flags due to other edge.
;
                MOVF    PORTB, 0            ; Move PortB value to the W register
                                            ;   This ends mismatch conditions
                MOVWF   TEMP                ; Need to save the PortB reading.
                XORWF   LASTPB, 1           ; XOR last PortB value with the new
                                            ;   PortB value.
CK_RB7          BTFSC   LASTPB, RB7         ; Did pin RB7 change
                CALL    RB7_CHG             ; RB7 changed and caused the interrupt
CK_RB6          BTFSC   LASTPB, RB6         ; Did pin RB6 change
                CALL    RB6_CHG             ; RB6 changed and caused the interrupt
CK_RB5          BTFSC   LASTPB, RB5         ; Did pin RB5 change
                CALL    RB5_CHG             ; RB5 changed and caused the interrupt
CK_RB4          BTFSC   LASTPB, RB4         ; Did pin RB4 change
                GOTO    RB4_CHG             ; RB4 changed and caused the interrupt
;
RB7_CHG             :                       ; Do task for INT on RB7
                    :                       ;
                RETURN
RB6_CHG         BTFSC   PORTB, RB6          ; Check for rising edge
                RETURN                      ; Falling edge, Ignore
                    :                       ; Do task for INT on RB6
                    :
                RETURN
RB5_CHG             :                       ; Do task for INT on RB5
                    :                       ;
                RETURN
RB4_CHG             :                       ; Do task for INT on RB4
                    :                       ;
CLR_RBINTF      MOVF    TEMP, 0             ; Move the PortB read value to the
                MOVWF   LASTPB              ;   register LASTPB
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                RETFIE                      ; Return from interrupt
;
OTHER_INT           :                       ; Do what you need to here
                    :
                RETFIE                      ; Return from interrupt
```

Author: Mark Palmer, Logic Products Division

© 1993 Microchip Technology Inc.

MICROCHIP

AN552

Implementing Wake-up on Key Stroke

SUMMARY

The PIC16CXX is ideally suited to interface directly to a Keypad application. Built in pull-up resistors and very low sleep current make it a very good candidate for battery powered remote operations and applications.

Performance:

Code Size: 64 words

RAM Locations Used: 0 bytes

INTRODUCTION

Microchip's PIC16CXX family of microcontrollers are ideally suited to directly interface to a keypad. The high 4-bits of PortB (RB4 - RB7) have internal pull-ups and can trigger a "change on port state" interrupt. This interrupt, if enabled, will wake the microcontroller from sleep. In most battery powered applications, a microcontroller is exercised when a key is pressed, e.g. in a remote keyless entry system. The life of the battery can be extended by using PIC16CXX microcontrollers. This can be done by putting the PIC16CXX microcontroller into sleep mode for most of the time and wake-up only when a key is pressed.

IMPLEMENTATION

Figure 1 depicts an application where four keys are connected to RB4 - RB7. Internal pull-ups are used to maintain a high level on these inputs. In this example, LEDs are connected to RB0 - RB3. When SW1 is pressed, LED1 is turned on and when SW2 is pressed, LED2 is turned on and so on. The PIC16CXX is normally in sleep mode with the "change on port state" interrupt enabled. When SW1 is pressed, RB4 goes low and triggers an interrupt. Since the PIC16CXX is in sleep, it first wakes up and starts executing code at the interrupt vector. Note that if the global interrupt is enabled, the program execution after an interrupt is at the interrupt vector, if the global interrupt is not enabled, the program starts executing right after the sleep instruction.

After waking up, a 20 - 40 msec. de-bounce delay is executed which checks the port for a key hit and, depending on which key is hit, its associated LED is turned on. The LEDs are used purely for demonstration purposes. In a remote keyless entry application, the remote code would be transmitted when the appropriate key is hit.

Figure 2 depicts a 4x4 keypad interface to the PIC16CXX. When using the PIC16CXX in a keypad application, the internal pull-ups on RB4 - RB7 can be enabled eliminating the need for external pull-up resistors. The series 100Ω resistors are used for ESD protection, and are recommended in keypad interface applications.

Author: Stan D'Souza, Logic Products Division

FIGURE 1 - 4 KEY INTERFACE TO PIC16CXX

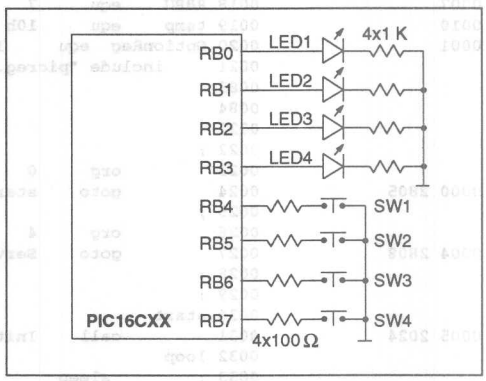
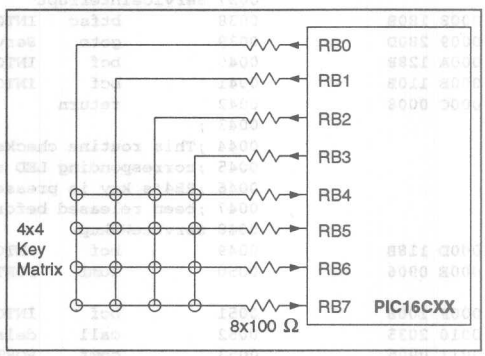


FIGURE 2 - 4X4 KEYPAD INTERFACE TO PIC16CXX



© 1994 Microchip Technology Inc.

```
MPASM 1.00 Released     WAKUP.ASM   7-15-1994  13:24:29                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;This program demonstrates the wake-up on Keystroke feature of the
                     0002 ;PIC16C71. Port B pins RB4 - RB7 can be configured as inputs with
internal
                     0003 ;pull up resistors, also the interrupt associated with the change on
input
                     0004 ;on RB4 - RB7 can be set up to wake the chip from sleep. If the
                     0005 ;global interrupt is enabled just before sleep, the program will vector
to
                     0006 ;the interrupt vector (0004). If not the chip will continue execution
                     0007 ;just after the next instruction following sleep.
                     0008 ;In this example code, the port B is initalized to input 4 push-buttons
at
                     0009 ;RB4 - RB7. RB0 - RB3 are configured to drive LEDs corresponding to
                     0010 ;which push-button is hit (LED on RB0 when RB4 is hit and so on).
                     0011 ;Sleep is executed. When any keys is hit, the processor wakes
                     0012 ;up and jumps to the interrupt vector. The corresponding LED is
                     0013 ;turned on and after the key is released, the whole process is re-
peated.
                     0014 ;
                     0015         LIST P=16C71, F=INHX8M
                     0016 ;
0002                 0017 z       equ     2
0007                 0018 RBPU    equ     7
0010                 0019 temp    equ     10h
0001                 0020 OptionReg  equ     1h
                     0021     include "picreg.equ"
                     0083
                     0084
                     0021
                     0022 ;
                     0023         org     0
0000 2805            0024         goto    start
                     0025 ;
                     0026         org     4
0004 2808            0027         goto    ServiceInterrupt
                     0028 ;
                     0029 ;
                     0030 start
0005 2024            0031         call    InitPortB          ;initalize port B
                     0032 loop
                     0033 ;        sleep                      ;sleep till key is hit
0006 0000            0034         nop
0007 2806            0035         goto    loop
                     0036 ;
                     0037 ServiceInterrupt
0008 180B            0038         btfsc   INTCON,RBIF        ;change on rb int?
0009 280D            0039         goto    ServiceWakup       ;yes then service
000A 128B            0040         bcf     INTCON,RTIE        ;clear RTCC int mask
000B 110B            0041         bcf     INTCON,RTIF        ;clear flag
000C 0008            0042         return
                     0043 ;
                     0044 ;This routine checks which keys is hit and lights up the
                     0045 ;corresponding LED associated with it. eg. RB0's LED when
                     0046 ;RB4's key is pressed. Finally it waits till all keys have
                     0047 ;been released before returning form the service routine.
                     0048 ServiceWakup
000D 118B            0049         bcf     INTCON,RBIE        ;clear mask
000E 0906            0050         comf    PORT_B,w           ;read PORT_B

000F 100B            0051         bcf     INTCON,RBIF        ;clear flag
0010 2035            0052         call    delay16            ;do de-bounce for 16mSecs
0011 0906            0053         comf    PORT_B,w           ;read port B again
0012 39F0            0054         andlw   B'11110000'        ;mask outputs
0013 0090            0055         movwf   temp               ;save in temp
```


© 1994 Microchip Technology Inc.

```
0014 0E10          0056         swapf   temp,w          ;switch low and high
0015 0086          0057         movwf   PORT_B          ;send as outputs.
0016 2018          0058         call    KeyRelease      ;check for key release
0017 0009          0059         retfie
                   0060 ;
                   0061 ;This sub-routine, waits till all key have been released
                   0062 ;In order to save power, the chip is in sleep mode till
                   0063 ;all keys are released.
                   0064 KeyRelease
0018 2035          0065         call    delay16         ;do debounce
0019 0906          0066         comf    PORT_B,w        ;read PORT_B
001A 100B          0067         bcf     INTCON,RBIF     ;clear flag
001B 158B          0068         bsf     INTCON,RBIE     ;enable mask
001C 39F0          0069         andlw   B'11110000'     ;clear outputs
001D 1903          0070         btfsc   STATUS,z        ;key still pressed?
001E 0008          0071         return                  ;no then return
001F 0063          0072         sleep                   ;else save power
0020 118B          0073         bcf     INTCON,RBIE     ;on wake up clear mask
0021 0906          0074         comf    PORT_B,w
0022 100B          0075         bcf     INTCON,RBIF     ;clear flag
0023 2818          0076         goto    KeyRelease      ;try again
                   0077 ;
                   0078 ;
                   0079 ;This sub-routine, initializes PortB.
                   0080 InitPortB
0024 1683          0081         bsf     STATUS,RP0      ;select bank 1
0025 3003          0082         movlw   B'00000011'     ;Port_A digital I/O
0026 0088          0083         movwf   ADCON1          ;               /
0027 3000          0084         movlw   0               ;
0028 0085          0085         movwf   PORT_A          ;set port a as outputs
0029 30F0          0086         movlw   B'11110000'     ;RB0-RB3 outputs
002A 0086          0087         movwf   PORT_B          ;RB4-RB7 inputs
002B 1381          0088         bcf     OptionReg,RBPU  ;enable pull up
002C 1283          0089         bcf     STATUS,RP0      ;select page 0
002D 0186          0090         clrf    PORT_B          ;init port B
002E 0185          0091         clrf    PORT_A          ;make port a all low
002F 1405          0092         bsf     PORT_A,0        ;make first bit high
0030 118B          0093         bcf     INTCON,RBIE     ;disable mask
0031 0806          0094         movf    PORT_B,w        ;read port
0032 100B          0095         bcf     INTCON,RBIF     ;clear flag
0033 158B          0096         bsf     INTCON,RBIE     ;enable mask
0034 0009          0097         retfie                  ;enable global and return
                   0098 ;
                   0099 ;delay16 waits for approx 16.4mSecs using RTCC interrupts
                   0100 ;fosc speed is 4Mhz.
                   0101 delay16
0035 1683          0102         bsf     STATUS,RP0      ;select page 1
0036 3007          0103         movlw   B'00000111'     ;fosc/256 -> RTCC
0037 0081          0104         movwf   OptionReg       ;       /
0038 1283          0105         bcf     STATUS,RP0      ;select page 0
0039 0181          0106         clrf    RTCC
003A 110B          0107         bcf     INTCON,RTIF     ;clear flag
003B 168B          0108         bsf     INTCON,RTIE     ;enable mask
                   0109 CheckAgain
003C 1D0B          0110         btfss   INTCON,RTIF     ;timer overflowed?
003D 283C          0111         goto    CheckAgain      ;no check again
003E 128B          0112         bcf     INTCON,RTIE     ;else clear mask
003F 110B          0113         bcf     INTCON,RTIF     ;clear flag
0040 0008          0114         return
                   0115 ;
                   0116         end
                   0117
                   0118
                   0119
                   0120
                   0121
                   0122
                   0123
                   0124
```

© 1994 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : X------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

Using the 8-Bit Parallel Slave Port

INTRODUCTION

The PIC16C64/C74 microcontrollers from Microchip Technology Inc., can be interfaced in a multi-microprocessor environment with ease using the built-in Parallel Slave Port. With their very high operating speeds (cycle times as low as 200ns with a clock rate of 20MHz), and an array of on-chip peripherals, they make ideal smart interfaces to the real world.

IMPLEMENTATION

PortD operates as an 8-bit wide parallel slave port, with PortE providing the control signals when bit PSPMODE (TRISE<4>) is set. In parallel slave mode, PortD is asynchronously readable and writable by the external world through the CS (RE2/CS), RD (RE0/RD), and WR (RE1/WR) control inputs.

In order to use the parallel slave port, the data direction bits in the TRISE register corresponding to RD, WR, and CE (TRISE<2:0>) must be configured as inputs (set=1).

The port pins are connected to two 8-bit latches, one for data output (from the PIC16CXX) and one for data input. The PIC16CXX sends data by writing to the output latch, and receives data by reading the input latch (note that the input and output latches are at the same address). In this mode the TRISD register is ignored, since the external device connected to the slave port controls the direction of data flow.

When the external device performs either a read or a write operation to the PIC16CXX, the interrupt flag, PSPIF (PIR1<7>), will be set and the processor interrupted if PSPIE (PIE1<7>) is set and interrupts are enabled (GIE and PEIE, (INTCON<7:6>) set). When the interrupt is serviced, PSPIF must be cleared by software.

The read-only status flag IBF, Input Buffer Full (TRISE<7>), is set if a received word is waiting to be read. IBF is cleared upon read of the input buffer latch. If another word is received prior to the first being read, status flag IBOV (TRISE<5>) is set. IBOV can be cleared by software.

The Output Buffer Full status bit, OBF (TRISE<6>), is set if a word written to PortD latch is waiting to be read by the external bus.

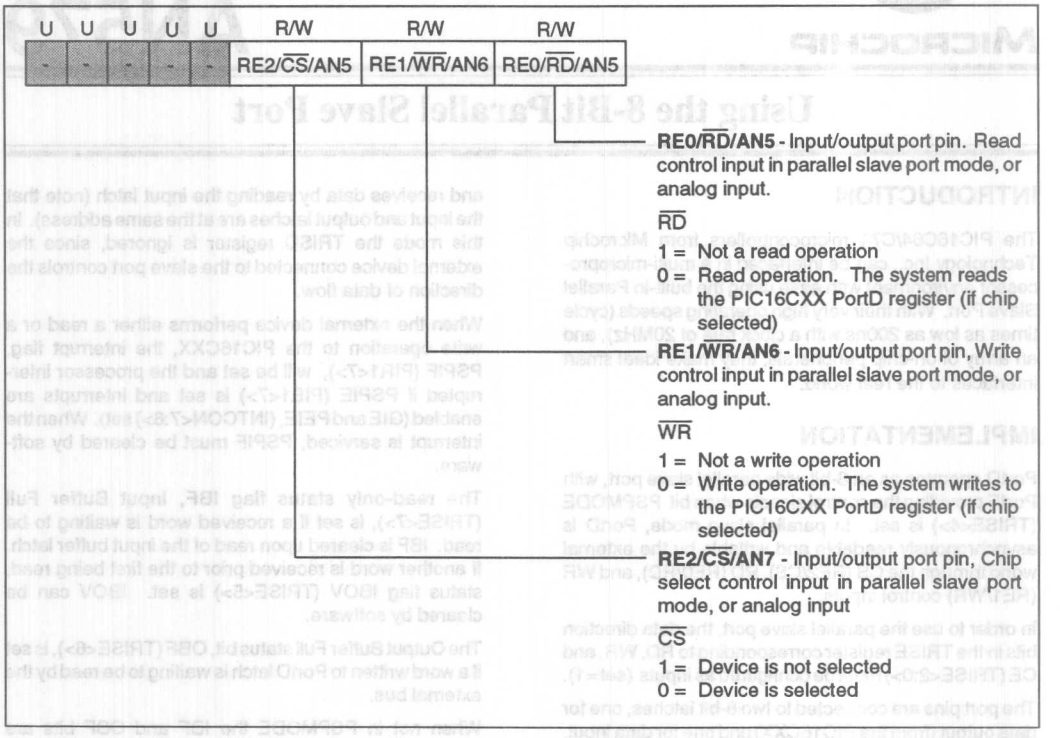
When not in PSPMODE the IBF and OBF bits are cleared. If the IBOV flag was previously set, however, it must be cleared by software.

FIGURE 1: SUMMARY OF PARALLEL SLAVE PORT REGISTERS

Register Name	Function	Address	Power-On Reset Value
PORTD	Parallel slave port Read/Write Data	08h	XXXX XXXX
TRISD	PortD data direction register	88h	1111 1111
PORTE	Parallel slave port Read/Write/Chip Select signals	09h	---- -XXX
TRISE	Control bits for PortD slave port	89h	0000 -111
INTCON	Global Interrupt Enable	0Bh	0000 000X
PIR1	Interrupt register (PSPIF bit)	0Ch	0000 0000
PIE1	Interrupt Enable register (PSPIE bit)	8Ch	0000 0000

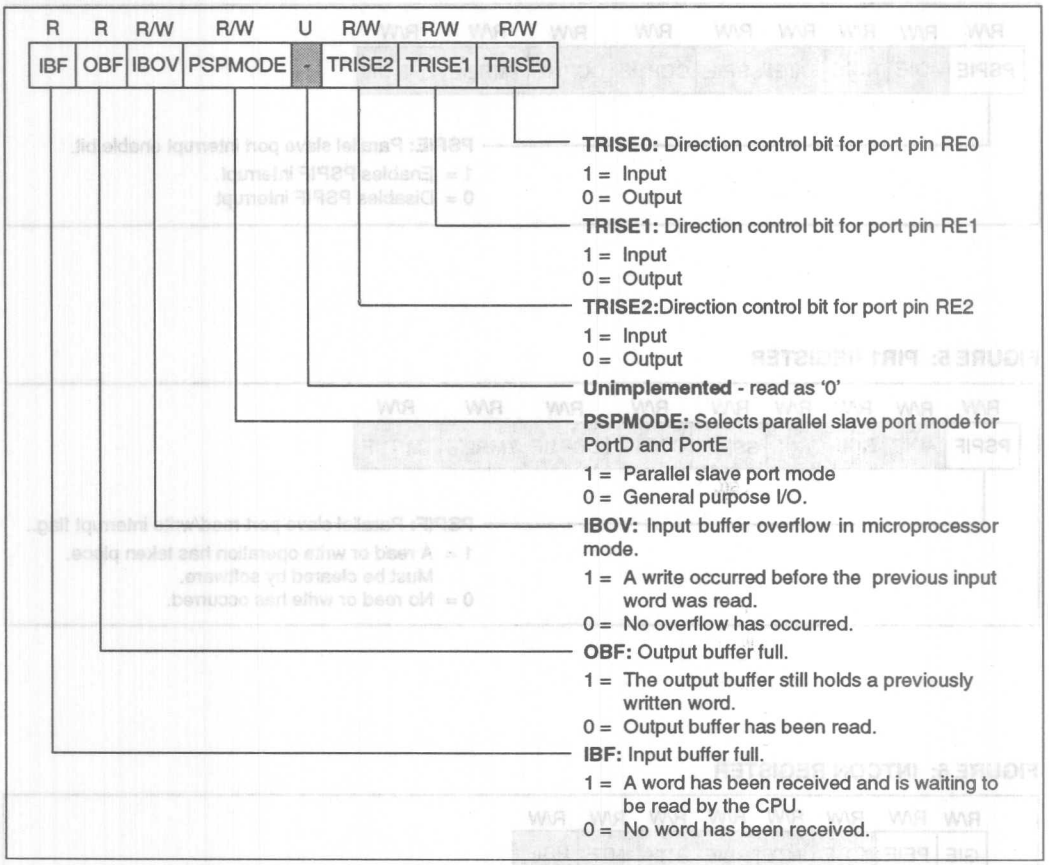
© 1994 Microchip Technology Inc.

FIGURE 2: PORT E FUNCTIONS



© 1994 Microchip Technology Inc.

FIGURE 3: TRISE REGISTER



3

© 1994 Microchip Technology Inc.

FIGURE 4: PIE1 REGISTER

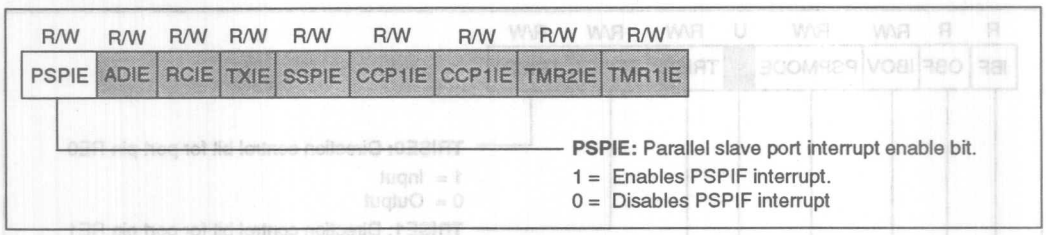


FIGURE 5: PIR1 REGISTER

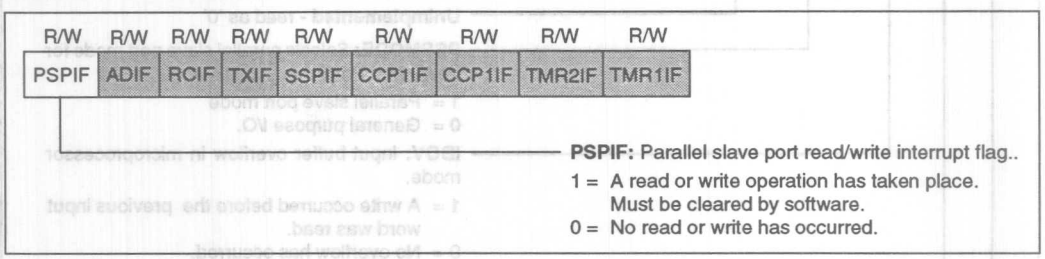
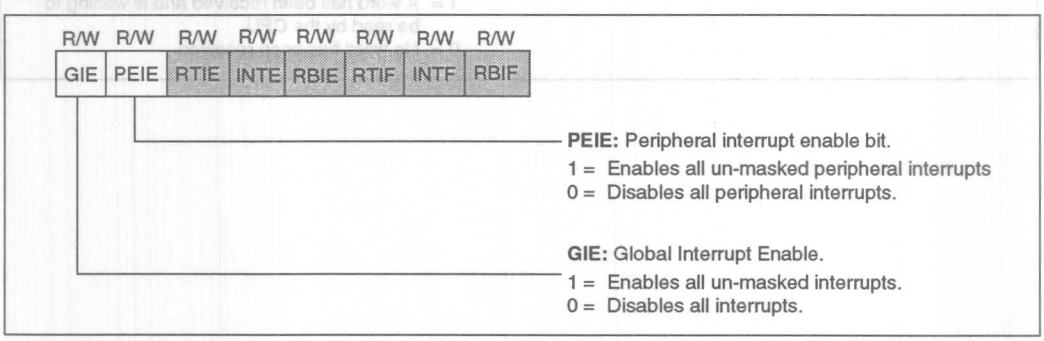


FIGURE 6: INTCON REGISTER



AUTHOR: Scott Fink, Logic Products Division

© 1994 Microchip Technology Inc.

```
;***************************************************************************
;* 16C64/74 Parallel Slave port
;*            This program demonstrates the Parallel Slave Port function of
;*            the PIC16C64/74.  The program is interrupt driven, when the PIC16CXX
;*            is either read from or written to, an interrupt is generated.  If the
;*            interrupt was caused by a read, a register is incremented, and
;*            the new count is placed in an output queue.  If the interrupt was
;*            caused by a write, the data is put on the Port B pins
;***************************************************************************
            list p=16c64,f=inhx8m
;
            include "c:\16C74.inc"

;Register definitions
FLAGREG     equ     20h             ;Flag bit register
OUTDATA     equ     21h             ;Output data
INDATA      equ     22h             ;Input data
COUNT       equ     23h             ;Count of times output register read

;Bit definitions for flag register
ERROR       equ     00h             ;Error flag bit
OUTRDY      equ     01h             ;Output data ready flag
INFULL      equ     02h             ;Input data received flag

            org     0000h           ;Reset Vector
            goto    Start

            org     0005h           ;Interrupt Vector
            goto    Service_Int

Start
            clrf    OUTDATA         ;Clear data registers
            clrf    INDATA
            bsf     STATUS,RP0      ;Select register page 1
            movlw   b'00010111'     ;Set RD, WR, and CS as
            movwf   TRIS_E          ; inputs, Enable Parallel Slave port
            movlw   0FFh
            movwf   TRIS_B          ;Set Port_B to all outputs
            movlw   b'10000000'     ;
            movwf   PIE1            ;Enable Parallel Slave Port interrupt
            bcf     STATUS,RP0      ;Select register page 0

            movf    OUTDATA,W       ;Set output Data in PORTD
            movwf   PORT_D
            movlw   b'11000000'     ;Set GIE, PEIE (enable interrupts)
            movwf   INTCON

Loop
            btfsc   FLAGREG,INFULL  ;Check if input data received
            goto    Checkout        ;No data ready, check output
            bcf     FLAGREG,INFULL  ;Clear input data ready flag
            movf    Indata,W        ;Get Input data
            movwf   PORT_B          ;Output input data to Port_B
Checkout
            btfsc   FLAGREG,OUTRDY  ;Check if data output already
            goto    Loop            ;Not output yet, loop
            incf    COUNT           ;Increment output data
            movf    COUNT,W         ;Get output data
            movwf   OUTDATA         ;Put data in output queue
            bsf     FLAGREG,OUTRDY  ;Set flag for interrupt routine
            goto    Loop
```


© 1994 Microchip Technology Inc.

```
;*************************************************************************
;*Interrupt Service Routine
;*            Inputs:    FLAGREG - Flag register to/from the main routine:
;*                                 Bit 1: OUTRDY - To Service_Int, indicates data
;*                                                 ready in output queue
;*                       OUTDATA - Output data queue
;*                       PIR1    - Interrupt flag register
;*                       TRIS_E  - Parallel slave port flag register
;*                       PORT_D  - Input data from slave port
;*
;*            Outputs:
;*                       PORT_D  - Output data to slave port
;*                       INDATA  - Input data queue
;*                       FLAGREG - Flag register to/from the main routine:
;*                                 Bit 0: ERROR - From Service_Int, indicates input
;*                                                buffer overflow
;*                                 Bit 2: INFULL- From Service_Int, indicates data
;*                                                received and in INDATA
;*************************************************************************

Service_Int
            btfss   PIR1,PSPIF      ;Test for Peripheral interrupt
            goto    Intout          ;Not a Peripheral interrupt, exit
            bcf     PIR1,PSPIF      ;Clear Peripheral interrupt
            bsf     STATUS,RP0      ;Select Page 1
            btfss   TRIS_E,IBF      ;Check if input data ready
            goto    Notinput        ;No input, check output
            bcf     STATUS,RP0      ;Input ready, select Page 0
            bsf     FLAGREG,INFULL  ;Set flag for main routine
            movf    PORT_D,W        ;Get input data
            movwf   INDATA          ;Put byte in input queue
Notinput
            btfsc   TRIS_E,OBF      ;Check if output data read
            goto    Intout          ;Not read, exit
            bcf     STATUS,RP0      ;Select Page 0
            btfss   FLAGREG,OUTRDY  ;Check if data in output queue
            goto    Intout          ;Output not read, exit
            movf    OUTDATA,W       ;Get data from queue
            movf    PORT_D          ;Put data in output buffer
            bcf     FLAGREG,OUTRDY  ;Clear flag for main routine
Intout
            bsf     STATUS,RP0      ;Select Page 1
            btfsc   TRIS_E,IBOV     ;Check input buffer overflow flag
            goto    Interror        ;If not clear, error
            bcf     STATUS,RP0      ;Select Page 0
            retfie                  ;Re-enable GIE and return
Interror
            bcf     STATUS,RP0      ;Select Page 0
            bsf     FLAGREG,ERROR   ;Set error flag for main routine
            retfie                  ;Re-enable GIE and return

            end
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN589

A PC-Based Development Programmer for the PIC16C84

Author: Robert Spur - Analog Design Specialist, Inc.

PROGRAMMING THE PIC16C84 MICROCONTROLLER

This application note describes the construction of a low cost serial programmer for the PIC16C84 microcontroller which is controlled using a PC with a parallel (Centronix printer) port. This programmer has the capability of programming the PIC16C84 microcontroller, and reading back internal data without removing the device form the target circuit.

This feature is very useful in applications where changes in program code or constants are necessary to compensate for other system features. For example, an embedded control system may have to compensate for variances in a mechanical actuator performance or loading. The basic program can be programmed and tested in design. The final program and control constants can be easily added later in the production phase without removing the microcontroller from the circuit.

Automatic software and performance upgrades can also be implemented with an in-system. Upon receiving new system software via disk or modem, a control processor with the included programming code could perform an in circuit reprogramming of other microcontrollers in the system.

This programmer can load program code, part configuration, and EEPROM data into the PIC16C84 part. In read back mode, it can verify all verify all data entries.

FUNCTION DESCRIPTION

The PIC16C84 microcontroller is put into programming mode by forcing a low logic level on the RB7 (pin 13) and RB6 (pin 12) while the MCLR (pin 4) is first brought low to reset the part, and then brought to the program/verify voltage of 12 to 14 volts. The MCLR pin remains at the program/verify voltage for the remainder of the programming or verification time.

After entering programming mode, RB7 is used to serially enter programming modes and data into the part. A high to low transition on RB6, the clock input, qualifies each bit of the data applied on RB7. The serial command-data format is specified in Figure 1.2.1.3 of the Microchip PIC16C84 Programming Specification (DS30189D). The first 6 bits form the command field, and the last 16 bits form the data field. Notice that the data field is composed of one zero starting bit, 14 actual data bits, and one zero stop bit. The increment address command, shown in Figure 1.2.1.5 (see PIC16C84 Data Sheet, DS30189D), is comprised of only the command field. Table 1.2.1.1 (see DS30189D) summarizes the available commands and command codes for serial programming mode.

The read mode is similar to programming mode with the exception that the data direction of RB7 is reversed after the 6-bit command to allow the requested data to be returned to the programmer. Figure 1.2.1.4 (see DS30189D) shows this sequence which starts by shifting the 6-bit command into the part. After the read command is issued, the programmer tri-states its buffer to allow the part to serially shift its internal data back to the programmer. The rising edge of RB6, the clock input, controls the data flow by sequentially shifting previously programmed or data bits from the part. The programmer qualifies this data on the falling edge of RB6. Notice that 16 clock cycles are necessary to shift out 14 data bits.

Accidental in circuit reprogramming is prevented during normal operation by the MCLR voltage which should never exceed the maximum circuit supply voltage of 6V DC and the logic levels of port bits RB7 and RB8.

After program/verification the MCLR pin is brought low to reset the target microcontroller and electrically released. The target circuit is then free to activate the MCLR signal. In the event MCLR is not forced by the target circuit, R4 (a 2K Ohm pull up resistor in the programmer) provides a high logic level on the target microcontroller which enables execution of its program independent of the programmer connection. Provisions should be made to prevent the target circuit from resetting the target microcontroller with MCLR or effecting the RB6 and RB6 during the programming process. In most cases this can be done without jumpers.

© 1994 Microchip Technology Inc.

DETAILED CIRCUIT DESCRIPTION

A logic high on PC parallel interface latch bit D4 turns on Q3 causing the MCLR pin to go low which places the target part in reset mode. The reset condition is then removed and the program/verify voltage is applied by placing a logic high on D3 and a logic low on D4 which turns off Q3 and turns on Q2 and Q1. Circuit protection of Q1 and Q3 is obtained from connecting the emitter of Q2 to latch bit D4 which prevents a simultaneous reset and program/verify voltage mode. Q2, a 2N3904, has a reverse emitter base break down voltage of 6 volts which will not be exceeded when 5 volt logic is used on the parallel interface.

The resistors R1, R2, R3, and the diode D1 provide a logic level interface to the analog circuitry. R4 provides a MCLR (master clear) pull up function during target circuit run mode. The programming voltage is supplied and adjusted by an external lab supply. This supply should have a current limit in the 100 ma range. 5 volts for U2 (LS244) is locally regulated from programming supply voltage by U1. R5 (750 ohm resistor) is connected to the regulator output to insures proper 5 volt regulation when the 13.5 volt programming voltage is applied through the pull up resistor R4.

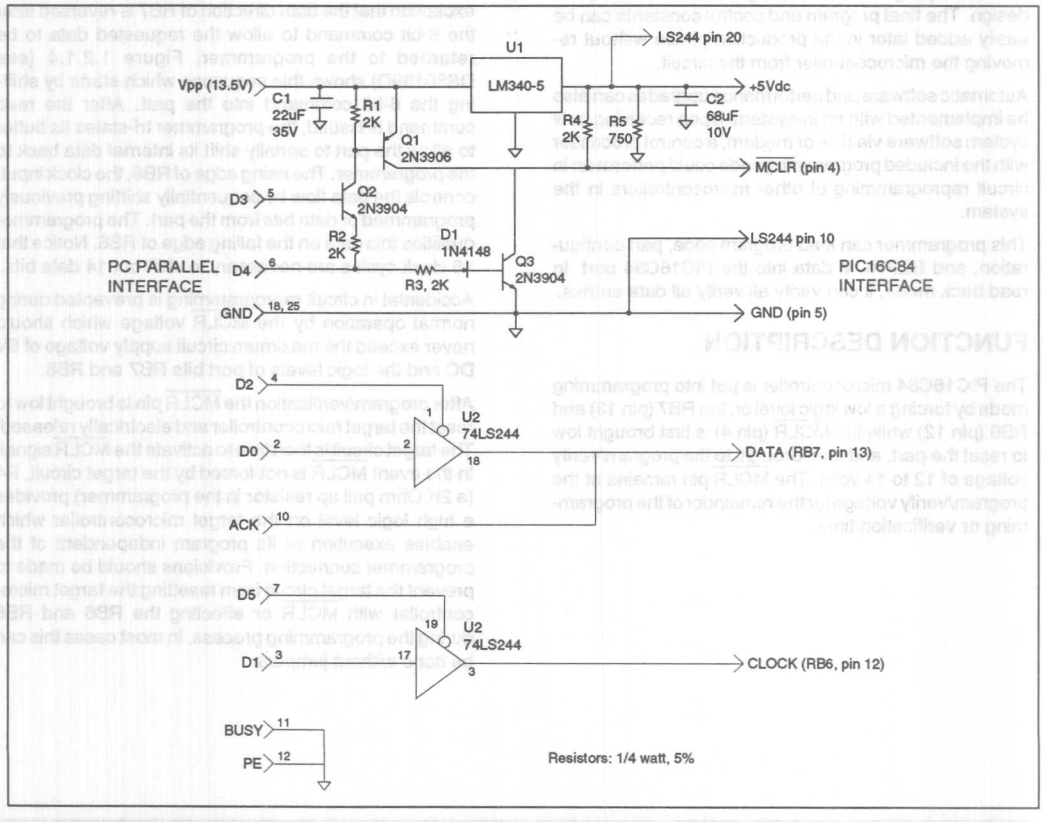
Data and clock are connected to the part via a tri-state buffer U2. PC parallel port interface bit D0 is used for data and port bit D1 is used for clock. During programming mode both clock and data buffers are enabled by port bits D2 and D5. During read mode, the data buffer is tri-stated via D2 and the printer data acknowledge signal line is used to receive verification data from the part.

After program/verification mode both the data and clock lines are tri-stated via D2 and D5 allowing the these lines to be used by the target circuit. This allows the programmer to remain physically, but not electrically connected to the target system.

An optional 5 volt line was included in the 3-foot programming interconnect cable for convince. Short interconnection leads and good grounding are always good construction practice.

To meet the programming verification specification, the target part's supply voltage should be first set to the maximum specified supply voltage and a program/data read back should be preformed. This process is then repeated at the lowest specified supply voltage.

FIGURE 1: PROGRAMMER SCHEMATIC



© 1994 Microchip Technology Inc.

SOFTWARE DESCRIPTION

The listed code provides a hardware-software interface to a standard PC parallel (Centronix) interface port. The code can be adapted to a microprocessor parallel interface port by substituting an output command for the "biosprint" command.

Control software can transfer the PIC16C84 program, configuration bits, and EEPROM data from a standard PROM interface file into the target system by reading the file and calling the function in Figure 2 using the appropriate command name in the definition table, and the data to be programmed. The command names are repeated here for reference.

LOAD_CONFIG	Sets PIC16C84 data pointer to configuration.
LOAD_DATA	Loads, but does not program, data.
READ_DATA	Reads data at current pointer location.
INC_ADDR	Increments PIC16C84 data pointer.
BEGIN_PROG	Programs data at current data pointer location.
PARALLEL_MODE	Puts PIC16C84 into parallel mode. (not used)
LOAD_DATA_DM	Loads EEPROM data.
READ_DATA_DM	Reads EEPROM data.

Function "int ser_pic16c84(<command>,<data [or 0]>) is called to preform command. Function returns internal data after read commands.

Do not forget to initiate the programming mode before programming, increment the addresses after each byte is programmed, and put the programmer in run mode after programming.

Designed by: Analog Design Specialist, Inc.
P.O. Box 26-0846
Littleton, CO 80126

3

EXAMPLE 1: PUT TARGET SYSTEM INTO PROGRAM MODE.

```
.. program code..
ser_pic16c84(PROGRAM_MODE,0);
.. program code..
```

EXAMPLE 2: READ DATA FROM THE TARGET SYSTEM

```
.. program code..
data = ser_pic16c84(READ_DATA,0); // read data
// transfers data from target part to variable "data".
.. more program code ..
```

EXAMPLE 3: PROGRAM DATA INTO THE TARGET SYSTEM

```
.. program code..
ser_pic16c84(LOAD_DATA,data);     // load data into target
ser_pic16c84(BEGIN_PROG,0);       // program loaded data
ser_pic16c84(INC_ADDR,0);         // increment to next address
 // transfers data from program variable "data" to target
    part.
.. more program code ..
```

EXAMPLE 4: PUT TARGET SYSTEM INTO RUN MODE

```
.. program code..
ser_pic16c84(RUN,0);
.. program code..
```

© 1994 Microchip Technology Inc.

```
//************************** FIGURE #2 ********************************
//**                                                                 **
//**   SERIAL PROGRAMMING ROUTINE FOR THE PIC16C84 MICROCONTROLLER   **
//**                                                                 **
//**              Analog Design Specialists                          **
//**                                                                 **
//*********************************************************************

//FUNCTION PROTOTYPE: int ser_pic16c84(int cmd, int data)

// cmd: LOAD_CONFIG   -> part configuration bits
//      LOAD_DATA     -> program data, write
//      READ_DATA     -> program data, read
//      INC_ADDR      -> increment to the next address (routine does not auto increment)
//      BEGIN_PROG    -> program a previously loaded program code or data
//      LOAD_DATA_DM  -> load EEPROM data regesters (BEGIN_PROG must folow)
//      READ_DATA_DM  -> read EEPROM data
//
// data: 1) 14 bits of program data or
//       2)  8 bits of EEPROM data (least significant 8 bits of int)

// Additional programmer commands (not part of PIC16C84 programming codes)
//
// cmd: RESET         -> provides 1 ms reset pulse to target system
//      PROGRAM_MODE  -> initializes PIC16C84 for programming
//      RUN           -> disconnects programmer from target system
//
// function returns:1) 14 or 8 bits read back data for read commands
//                  2) zero                          for write data commands
//                  3) PIC_PROG_EROR = -1 for programming function errors
//                  4) PROGMR_ERROR  = -2 for programmer function errors


#include <bios.h>

#define LOAD_CONFIG    0
#define LOAD_DATA      2
#define READ_DATA      4
#define INC_ADDR       6
#define BEGIN_PROG     8
#define PARALLEL_MODE 10  // not used
#define LOAD_DATA_DM   3
#define READ_DATA_DM   5
#define MAX_PIC_CMD   63  // division between pic and programmer commands

#define RESET         64  // external reset command, not neded for programming
#define PROGRAM_MODE  65  // initialize program mode
#define RUN           66  // electrically disconnect programmer

#define PIC_PROG_EROR -1
#define PROGMR_ERROR  -2

#define PTR            0  // use device #0

// parallel port bits
//      d0: data output to part to be programmed
//      d1: programming clock
//      d2: data dirrection, 0= enabel tri state buf -> send data to part
//      d3: Vpp control 1= turn on Vpp
//      d4: ~MCLR =0, 1 = reset device with MCLR line
//      d5: clock line tri state control, 0 = enable clock line

int ser_pic16c84(int cmd, int data)          // custom interface for pic 16c84
  {
  int i, s_cmd;

  if(cmd <=MAX_PIC_CMD)                      // all programming modes
    {
    biosprint(0,8,PTR);                      // set bits 001000, output mode, clock & data low
```


© 1994 Microchip Technology Inc.

```
    s_cmd = cmd;                                    // retain command "cmd"
    for (i=0;i<6;i++)                               // output 6 bits of command
      {
      biosprint(0,(s_cmd&0x1) +2+8,PTR);            // set bits 001010, clock hi
      biosprint(0,(s_cmd&0x1)   +8,PTR);            // set bits 001000, clock low
      s_cmd >>=1;
      }

    if((cmd ==INC_ADDR)||(cmd ==PARALLEL_MODE)      // command only, no data cycle
      return 0;

    else if(cmd ==BEGIN_PROG)                       // program command only, no data cycle
      {
      delay(10);                                    // 10 ms PIC programming time
      return 0;
      }

    else if((cmd ==LOAD_DATA)||(cmd ==LOAD_DATA_DM)||(cmd ==LOAD_CONFIG)) // output 14 bits of
data
      {
      for (i=200;i;i--) ;                           // delay between command & data
      biosprint(0,2+8,PTR);                         // set bits 001010, clock hi; leading bit
      biosprint(0,  8,PTR);                         // set bits 001000, clock low

      for (i=0;i<14;i++)                            // 14 data bits, lsb first
        {
        biosprint(0,(data&0x1) +2+8,PTR);           // set bits 001010,  clock hi
        biosprint(0,(data&0x1)   +8,PTR);           // set bits 001000,  clock low
        data >>=1;
        }
      biosprint(0,2+8,PTR);                         // set bits 001010, clock hi; trailing bit

   // ***************** Analog Design Specialists ******************


      biosprint(0,  8,PTR);                         // set bits 001000, clock low

      return 0;
      }

    else if((cmd ==READ_DATA)||(cmd ==READ_DATA_DM)) //read 14 bits from part, lsb first
      {
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low, tri state data
                                                          buffer
      for (i=200;i;i--) ;                           // delay between command & data
      biosprint(0,2+4+8,PTR);                       // set bits 001110, clock hi, leading bit
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low

      data =0;
      for (i=0;i<14;i++)                            // input 14 bits of data, lsb first
        {
        data >>=1;                                  // shift data for next input bit
        biosprint(0,2+4+8,PTR);                     // set bits 001110,  clock hi
        biosprint(0,  4+8,PTR);                     // set bits 001100,  clock low
        if(!(biosprint(2,0,0)&0x40)) data += 0x2000; //use printer acknowledge line for input,
                                                          data lsb first
        }
      biosprint(0,2+4+8,PTR);                       // set bits 001110, clock hi, trailing bit
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low
      return data;
      }

    else return PIC_PROG_EROR;                      // programmer error

    }
  else if(cmd == RESET)                             // reset device
    {
    biosprint(0,32+16+4,PTR);                       // set bits 110100, MCLR = low (reset
                                                          PIC16C84), programmer not conected
```

© 1994 Microchip Technology Inc.

```
  delay(1);                                   // 1ms delay
  biosprint(0,32    +4,PTR);                  // set bits 100100, MCLR = high
  return 0;
  }

else if(cmd ==PROGRAM_MODE)                   // enter program mode
  {
  biosprint(0,32+16+4,PTR);                   // set bits 110100, Vpp off, MCLR = low
                                                 (reset PIC16C84)
  delay(10);                                  // 10 ms, allow programming voltage to
                                                 stabelize

  biosprint(0,8,PTR);                         // set bits 001000, Vpp on , MCLR = 13.5
                                                 volts, clock & data connected
  delay(10);                                  // 10 ms, allow programming voltage to
                                                 stablize

  return 0;
  }

else if(cmd ==RUN)                            // disconects programmer from device
  {
  biosprint(0,32+4,PTR);                      // set bits 100100
  return 0;
  }
else return PROGMR_ERROR;                     // command error
}
```


© 1994 Microchip Technology Inc.

Using the CCP Modules

This application note discusses the operation of a Capture Compare and PWM (CCP) module, and the interaction of multiple CCP modules with the timer resources.

The Capture Compare and PWM (CCP) module is software programmable to operate in one of three modes:

1. A Capture input
2. A Compare output
3. A Pulse Width Modulation (PWM) output

For the CCP module to function, Timer resources must be used in conjunction with the CCP module. The desired CCP mode of operation determines which timer resources are required. Table 1 shows the CCP mode with the corresponding timer resource required. Both the Capture and Compare modes require that Timer 1 be operating in timer mode or synchronized counter mode.

Note: Capture and Compare modes may not operate if Timer1 is operated in asynchronous counter mode.

TABLE 1: CCP MODE - TIMER RESOURCE

CCP Mode	Timer Resource
Capture	Timer 1
Compare	Timer 1
PWM	Timer 2

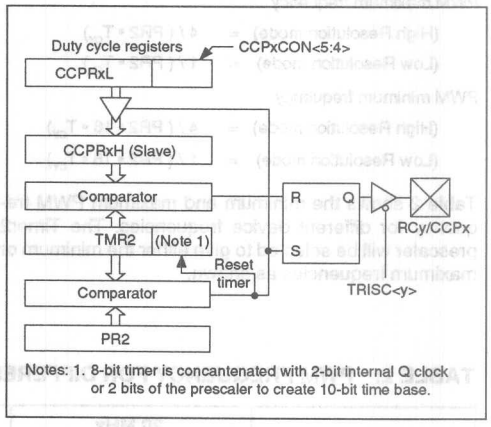
CCP OPERATION

The following three sections discuss the operation of the CCP module in each of its modes of operation. There is a simple example program for each mode of operation. The software example for the capture mode, also uses a second CCP module in compare mode to generate the signal to capture.

PWM Mode

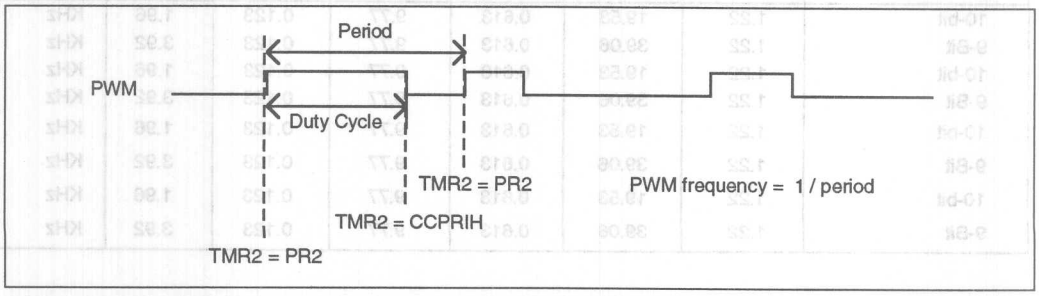
A Pulse Width Modulation output (shown in Figure 1) is a signal that has a timebase (period) and a time that the output stays high (duty cycle). The period is the duration after which the PWM rising edge repeats itself. The resolution of the PWM output is the granularity with which the duty cycle can be varied. The frequency of a PWM is simply the inverse of the period (1 / period).

FIGURE 2: PWM MODE BLOCK DIAGRAM



Notes: 1. 8-bit timer is concantenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time base.

FIGURE 1: PWM OUTPUT



© 1994 Microchip Technology Inc.

Each CCP module can support one Pulse Width Modulation (PWM) output signal, with minimal software overhead. This PWM signal can attain a resolution of up to 10-bits, from the 8-bit Timer 2 module. This gives 1024 steps of variance from an 8-bit overflow counter. This gives a maximum accuracy of Tosc (50 ns, when the device is operated at 20 MHz). Figure 2 shows a block diagram of the CCP module in PWM mode. When the Timer 2 overflows (timer = Period Register), the value in the duty cycle registers (CCPRxL:CCPRxCON<5:4>) is latched into the 10-bit slave latch. A new duty cycle value can be loaded into the duty cycle register(s) at any time, but is only latched into the slave latch when Timer 2 = Timer 2 Period Register (PR2).

The period of Timer 2 (and PWM) is determined by the frequency of the device, the Timer 2 prescaler value (1, 4 or 16), and the Timer 2 Period Register. Equation 1 shows the calculation of the PWM period, duty cycle, and the minimum and maximum frequncies.

EQUATION 1: PWM PERIOD, DUTY CYCLE, AND FREQUNCIES

PWM Period = $[(PR2) + 1) \bullet 4\ T_{osc}$ • (Timer 2 prescale value)

PWM Duty Cycle = $[CCPRxL:CCPRxCON<5:4>] \bullet 4\ T_{osc}$ • (Timer 2 prescale value)

PWM maximum frequency

(High Resolution mode) = $4 / (PR2 \bullet T_{CY})$

(Low Resolution mode) = $1 / (PR2 \bullet T_{CY})$

PWM minimum frequency

(High Resolution mode) = $4 / (PR2 \bullet 16 \bullet T_{CY})$

(Low Resolution mode) = $1 / (PR2 \bullet 16 \bullet T_{CY})$

Table 2 shows the minimum and maximum PWM frequency for different device frequencies. The Timer2 prescaler will be selected to give either the minimum or maximum frequencies as shown.

Appendix A is a program which generates up to a 10-bit PWM output. The PWM period and duty cycle are updated after the overflow of Timer1. Upon the overflow of Timer1, ports A, B and D are read. The 10-bit duty cycle is specified by the value on PORTB:PORTA<1:0>, while the period is specified by the value on PORTD. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

Since the PWM duty cycle is double buffered, the duty cycle registers are only loaded when there is sufficient time to complete the update the 10-bit value before the Timer2 = PR2 match occurs. After the duty cycle has been updated and the Timer2 = PR2 match has occurred, the period (stored in the PR2 register) is updated. The operation of the CCP module in PWM mode is similar to the PIC17C42's PWM. Additional concepts of PWM operation can be found in Application Notes AN564 and AN539.

TABLE 2: PWM FREQUENCY FOR DIFFERENT DEVICE FREQUENCIES

	20 MHz		10 MHz		2 MHz		
PWM Resolution	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz

© 1994 Microchip Technology Inc.

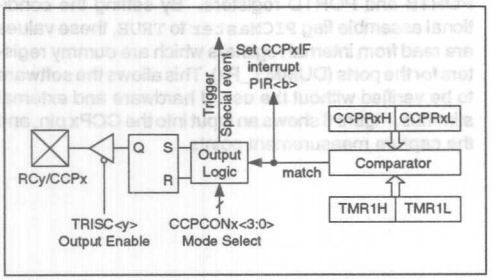
Compare Mode

In compare mode, the 16-bit value of Timer1 is compared to the CCPRxH:CCPRxL registers. When these registers match, the S/W configured event occurs on the CCPx pin. The events that can be S/W selected are:

- Clear CCPx pin on match
- Set CCPx pin on match
- Generate S/W interrupt (CCPx pin unchanged)
- Trigger special event (CCPx pin unchanged)
  - CCP1 clears Timer1
  - CCP2 clears Timer1 and sets the A/D's GO bit

The CCPxM<3:0> control bits, in register CCPxCON, configures the operation of the CCP module. The compare function must have the data direction of the CCPx pin configured as an output, if the compare event is to control the state of the CCPx pin.

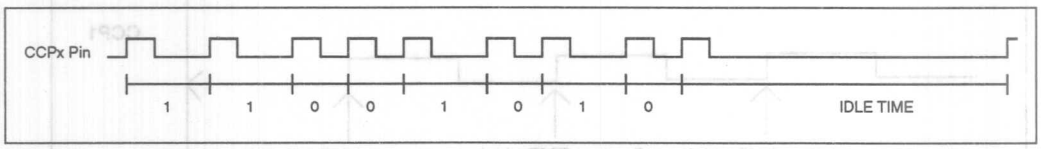
FIGURE 3: COMPARE MODE BLOCK DIAGRAM



When the CCP module is in the OFF state (CCPxM<3:0> = 0h), the CCPx output latch is forced to a low level, though the level on the CCPx pin will be determined by the value in the data latch of the port. Figure 3 shows the block diagram of the CCP module in Compare mode.

Appendix B is a program which uses the CCP module to transmit a pulse train dependent on the data byte. Timer1 is used as a free running timer, with each "new" compare value being an offset added to the present CCP compare latch value. The data is transmitted every 600 µs. Each data bit has a sync pulse (High level) of 8.8 µs. Then the data is transmitted as a low pulse. The time duration of the low pulse determines the value of the data bit. A '0' bit is low for 18.8 µs while a '1' bit is low for 37.6 µs. After the last data bit has been transmitted, another sync pulse is transmitted and the output remains low (idle time) until the 600 µs data period has completed. An example of the pulse train for the a data byte of CAh is shown in Figure 4, and has an idle time of 224 µs. These pulse times are based off the device operational frequency. The program header file, COMP.H, calculates the values to loaded into the compare registers from the specified Device_freq. The data to be transmitted is read from PORTB, during the idle time. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

FIGURE 4: TRANSMIT PULSE TRAIN (DATA = 0X0CA)



© 1994 Microchip Technology Inc.

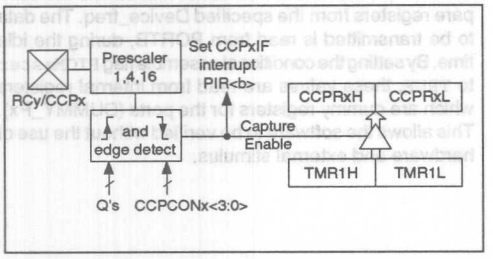
Capture Mode

In capture mode, the 16-bit value of Timer1 is latched into the CCPRxH:CCPRxL registers, when the S/W configured event occurs on the CCPx pin. The events that can cause a capture are:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

The CCPxM<3:0> control bits, in register CCPxCON, configures the operation of the CCP module. The capture function works regardless of the data direction of the CCPx pin (input or output). With the CCPx pin is configured as an output, a write to the CCPx pin (in PORTC) will cause a capture when the capture requirement is met.

FIGURE 5: CAPTURE MODE BLOCK DIAGRAM

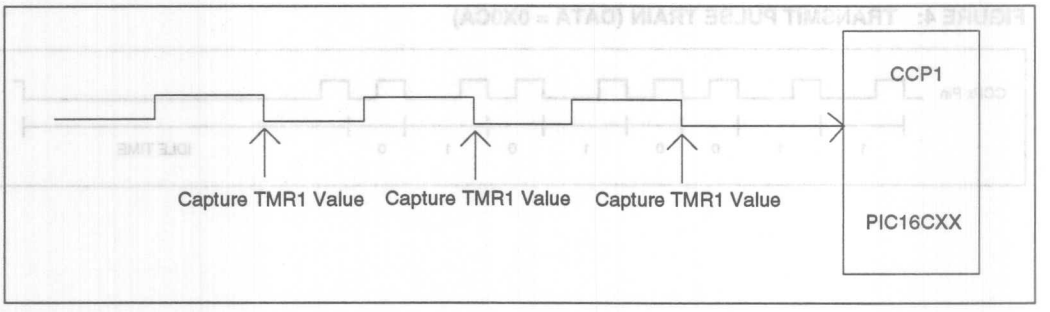


The changing of the capture mode, via the CCPxM<3:0> bits, may cause the CCPxIF bit to be set. This "false" interrupt should be cleared (ignored) after changing between capture modes. The CCP prescaler is only cleared by configuring the CCP module into the OFF state (CCPxM<3:0> = 0h). Figure 5 shows the block diagram of the CCP module in Capture mode. The utilization of the CCP module in capture mode is similar to the PIC17C42's capture. Additional concepts of capture operation can be found in Application Note AN545.

Appendix C is a program which implements a 16-bit capture from a free running timer (TMR1). The capture event is configured as each rising edge. The 16-bit capture value is the "new" 16-bit capture value minus the "old" 16-bit capture value. If the time between captures is greater than 2^{16} Timer1 increments, an invalid result will occur. This invalid result is not indicated by the software. After the capture period result is calculated, the "new" capture value is loaded into the "old" register.

The waveform that is captured is generated from a second CCP module in compare mode. The value that is loaded in to the CCPR2H:CCPR2L is read from the PORTB and PORTD registers. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus. Figure 6 shows an input into the CCPx pin, and the capture measurement points.

FIGURE 6: EXAMPLE CAPTURE WAVE FORM



© 1994 Microchip Technology Inc.

INTERACTION OF CCP MODULES

Due to the modularity of the PIC16CXX peripherals, future devices with two or more CCP modules on a device are possible. Each CCP module operates independently from the others, though their interaction with the timer resources must be taken into account.

When two or more CCP modules exist on a device, there can be an interaction between the CCP modules. This interaction is shown in Table 3. These interactions do NOT include any interaction (S/W) caused by the main program nor the interrupt service routines of the CCP sources.

Interaction of Two Capture Modes

When two CCP modules are in a Capture mode, Timer1 is the timebase for both captures. This means that they will have the same capture resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the device.

Interaction of One Capture Mode and One Compare Mode

When one CCP module is in a Capture mode and a second CCP module is in Compare mode, Timer1 is the timebase for both the captures and the compare. This means that the capture and the compare will have the same resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Also, care must be taken in that the compare can be configured to clear TMR1 (when in special Trigger mode). Care must be taken in system design to ensure that this clearing of the TMR1 does not have any negative impact on the capture function.

Interaction of Two Compare Modes

When two CCP modules are in a Compare mode, Timer1 is the timebase for both compares. This means that they will have the same compare resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Since the compare modules can be configured to clear TMR1 (when in special Trigger mode), care must be taken in system design to ensure that this clearing of the TMR1 does not have any negative impact on the compare function. If both compares are configured with a special trigger, which clears the TMR1, then the compare register that is closest to (but greater than) the TMR1 value is the compare value that will reset TMR1. Example 1 shows a possible case.

TABLE 3: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 timebase.
Capture	Compare	The compare could be configured for trigger special event, which clears TMR1.
Compare	Compare	The compare(s) could be configured for trigger special event, which clears TMR1.
PWM	PWM	The PWMs will have the same frequency, and update rate (TMR2 interrupt).
PWM	Capture	None
PWM	Compare	None

© 1994 Microchip Technology Inc.

EXAMPLE 1:

ACTION	TIMER 1 STATE	COMMENT
CCPR1H:CCPR1L = 0x0465	0x????	
CCP1CON = 0x?B	0x????	CCP1 in Compare - Special
:		Trigger Mode
:	0x0232	
:		
CCPR2H:CCPR2L = 0x0165	0x0333	
CCP2CON = 0x?B	0x0334	CCP2 in Compare - Special
:		Trigger Mode
	0x0465	CCP1 resets TMR1 and CCP1 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		

Interaction of Two PWM Modes

When two CCP modules are in a PWM mode, Timer2 is the timebase for both PWM outputs. This means that they will have the same PWM frequency and update rates, as determined by the TMR2 prescaler and frequency of the device. The resolution of the two PWMs may be different, since each CCP module has its own CCPxX:CCPxY bits for high resolution mode. These bits are found in the CCPxCON<5:4> register.

CONCLUSION

The Capture / Compare / PWM modules offer enormous flexibility in the use of the device timer resources. As with all resources, care must be taken to ensure that no adverse system complications can occur with the interaction between multiple CCP modules. The programs for simple operation of the various CCP modes should be a good foundation for modifications to suite your particular needs.

Written by: Mark Palmer - Sr. Application Engineer
Logic Products Division

© 1994 Microchip Technology Inc.

```
MPASM 01.01 Released    PWM_1.ASM  7-13-1994  14:26:3        PAGE 1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001          LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*************************************************************************
                     0004 ;
                     0005 ; This program outputs a PWM signal on the CCP1 pin. The duty cycle and period
                     0006 ; of the PWM is read every time TMR1 overflows.
                     0007 ;     PERIOD     = PORTB
                     0008 ;     DUTY CYCLE = PORTD and PORTE<1:0>
                     0009 ;
                     0010 ; The prescaler of TMR2 is selected by the state of PORTA<1:0> after reset
                     0011 ;      RA1:RA0    Prescaler multiplies Tcyc by
                     0012 ;        0 0       1
                     0013 ;        0 1       4
                     0014 ;        1 x      16
                     0015 ;
                     0016 ;
                     0017 ;        Program = PWM_1.ASM
                     0018 ;        Revision Date:   7-13-94
                     0019 ;
                     0020 ;*************************************************************************
                     0021 ;
                     0022 ;
                     0023 ; HARDWARE SETUP
                     0024 ;     PORTA<1:0>    -  Prescaler to TMR2, read only after reset
                     0025 ;     PORTB         -  Period of PWM
                     0026 ;     PORTD         -  Duty Cycle high of PWM (8-bits)
                     0027 ;     PORTE<1:0>    -  Duty Cycle low of PWM (2-bits)
                     0028 ;
                     0029 ;
                     0030           INCLUDE <C74_reg.h>
                     0247
                     0030
                     0031           INCLUDE <PWM.h>
                     0047
                     0031
                     0032 ;
0001                 0033 PICMaster       EQU     TRUE            ; A Debugging Flag
0001                 0034 Debug           EQU     TRUE            ; A Debugging Flag
0001                 0035 Debug_PU        EQU     TRUE            ; A Debugging Flag
                     0036 ;
                     0037 ;
                     0038 ; Reset address. Determine type of RESET
```


© 1994 Microchip Technology Inc.

```
                        0039 ;
                        0040         org     RESET_V                 ; RESET vector location
0000 1683               0041 RESET       BSF     STATUS, RP0         ; Bank 1
0001 188E               0042             BTFSC   PCON, POR           ; Power-up reset?
0002 2832               0043             GOTO    START               ; YES
0003 285C               0044             GOTO    OTHER_RESET         ; NO, a WDT or MCLR reset
                        0045 ;
                        0046 ; This is the Periperal Interrupt routine. Need to determine the type
                        0047 ; of interrupt that occurred. The following interrupts are enabled:
                        0048 ;   1.  CCP Capture Occured
                        0049 ;
                        0051         org     ISR_V                   ; Interrupt vector location
                        0052 PER_INT_V
0004 1283               0053             BCF     STATUS, RP0         ; Bank 0
0005 180C               0054             BTFSC   PIR1, TMR1IF        ; TMR1 Overflow Interrupt occured?
0006 280D               0055             GOTO    T1OVFL              ; YES, Service the TMR1 Interrupt
                        0056 ERROR1                                  ; NO, Error Condition - Unknown Interrupt
0007 1505               0057             BSF     PORTA, 2            ; Toggle a PORT pin
0008 1105               0058             BCF     PORTA, 2
0009 2807               0059             GOTO    ERROR1
                        0060 ;
                        0061 ERROR2                                  ; NO, Error Condition - Unknown Interrupt
000A 1585               0062             BSF     PORTA, 3            ; Toggle a PORT pin
000B 1185               0063             BCF     PORTA, 3
000C 280A               0064             GOTO    ERROR2
                        0065 ;
                        0066 T1OVFL
000D 100C               0067             BCF     PIR1, TMR1IF        ; Clear T1 Overflow Interrupt Flag
                        0068         if (PICMaster )
000E 0853               0069             MOVF    DUMMY_PD, W         ;
                        0070         else
                        0071             MOVF    PORTD, W            ;
                        0072         endif
000F 00D5               0073             MOVWF   DC_HI               ;
                        0074         if (PICMaster )
0010 0854               0075             MOVF    DUMMY_PE, W         ;
                        0076         else
                        0077             MOVF    PORTE, W            ;
                        0078         endif
0011 00D6               0079             MOVWF   DC_LO               ;
                        0080         if (PICMaster )
0012 0851               0081             MOVF    DUMMY_PB, W         ;
                        0082         else
                        0083             MOVF    PORTB, W            ;
                        0084         endif
0013 1683               0085             BSF     STATUS, RP0         ; Bank 1
0014 00A0               0086             MOVWF   T2_PERIOD           ;
0015 1283               0087             BCF     STATUS, RP0         ; Bank 0
```


© 1994 Microchip Technology Inc.

```
                        0088 ;
                        0089 WAIT_DC
0016 0811               0090            MOVF    TMR2, W         ; Read present TMR2 register value
0017 0212               0091            SUBWF   PR2, W          ; How close is the timer to rolling over
0018 390F               0092            ANDLW   0x0F            ; Does this make it zero?
0019 1903               0093            BTFSC   STATUS, Z       ; If Z is set, near rollover
001A 2816               0094            GOTO    WAIT_DC         ; loop until rolled over
001B 0855               0095            MOVF    DC_HI, W        ; else losd the duty cycle values
001C 0095               0096            MOVWF   CCPR1L          ; Load DC high
001D 300F               0097            MOVLW   0x0F            ;
001E 0597               0098            ANDWF   CCP1CON, F      ; Set the DC low bits
001F 18D6               0099            BTFSC   DC_LO, 1        ;
0020 1697               0100            BSF     CCP1CON, CCP1X  ;
0021 1856               0101            BTFSC   DC_LO, 0        ;
0022 1617               0102            BSF     CCP1CON, CCP1Y  ;
0023 108C               0103            BCF     PIR1, TMR2IF    ; Clear the TRM2 = PR2 flag
                        0104 ;
                        0105 WAIT_PR
0024 1C8C               0106            BTFSS   PIR1, TMR2IF    ; LOOP waiting for TRM2 = PR2
0025 2824               0107            GOTO    WAIT_PR         ; Need to wait until TMR2 = PR2 so that
                        0108                                    ;    Duty Cycle is latched
0026 1683               0109            BSF     STATUS, RP0     ; Bank 1
0027 300F               0110            MOVLW   0x0F            ; Load TMR2 period with minimum value Fh
0028 0092               0111            MOVWF   PR2             ;
0029 30F0               0112            MOVLW   0xF0            ;
002A 0520               0113            ANDWF   T2_PERIOD, W    ; Determine if period needs to be greater
002B 1903               0114            BTFSC   STATUS, Z       ;
002C 2830               0115            GOTO    NO_OFFSET       ; NO, Period is the minimum
                        0116 PR_OFFSET
002D 300F               0117            MOVLW   0x0F            ; Yes, calculate additional offset
002E 0220               0118            SUBWF   T2_PERIOD, W    ;
002F 0792               0119            ADDWF   PR2, F          ; ADD Period offset
                        0120 ;
                        0121 NO_OFFSET
0030 1283               0122            BCF     STATUS, RP0     ; Bank 0
0031 0009               0123            RETFIE                  ; Return / Enable Global Interrupts
                        0124 ;
                        0126 ;
                        0127 ;*************************************************************************
                        0128 ;*****      Start program here, Power-On Reset occurred.
                        0129 ;*************************************************************************
                        0130 ;
                        0131 START                              ; POWER_ON Reset (Beginning of program)
0032 1283               0132            BCF     STATUS, RP0     ; Bank 0
0033 018F               0133            CLRF    TMR1H           ;
0034 018E               0134            CLRF    TMR1L           ;
                        0135 ;
                        0136 MCLR_RESET                         ; A Master Clear Reset
```

© 1994 Microchip Technology Inc.

```
0035 0183          0137              CLRF    STATUS          ; Do initialization (Bank 0)
0036 018B          0138              CLRF    INTCON
0037 018C          0139              CLRF    PIR1
0038 1683          0140              BSF     STATUS, RP0     ; Bank 1
0039 3080          0141              MOVLW   0x80            ;
003A 0081          0142              MOVWF   OPTION_R        ;
003B 018C          0143              CLRF    PIE1            ; Disable all peripheral interrupts
003C 30FF          0144              MOVLW   0xFF            ;
003D 009F          0145              MOVWF   ADCON1          ; Port A is Digital.
                   0146 ;
                   0147 ;
003E 1283          0148              BCF     STATUS, RP0     ; Bank 0
003F 0185          0149              CLRF    PORTA           ; ALL PORT output should output Low.
0040 0186          0150              CLRF    PORTB
0041 0187          0151              CLRF    PORTC
0042 0188          0152              CLRF    PORTD
0043 0189          0153              CLRF    PORTE
                   0154 ;
0044 1683          0155              BSF     STATUS, RP0     ; Select Bank 1
0045 30FF          0156              MOVLW   0xFF            ;
0046 0085          0157              MOVWF   TRISA           ; RA5 -  0 inputs
0047 0086          0158              MOVWF   TRISB           ; RB7 - 0 inputs
0048 0187          0159              CLRF    TRISC           ; RC Port are outputs
0049 0088          0160              MOVWF   TRISD           ; RD Port are inputs
004A 0089          0161              MOVWF   TRISE           ; RE Port are inputs
004B 0092          0162              MOVWF   PR2             ; Default PWM period
004C 140C          0163              BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
004D 1283          0164              BCF     STATUS, RP0     ; Select Bank 0
                   0165 ;
004E 300C          0166              MOVLW   0X0C            ; CCP module is in
004F 0097          0167              MOVWF   CCP1CON         ; PWM output mode
                   0168 ;
                   0169 ; Initialize the Special Function Registers (SFR) interrupts
                   0170 ;
0050 018C          0171              CLRF    PIR1            ;
0051 0190          0172              CLRF    T1CON           ;
0052 0192          0173              CLRF    T2CON           ;
                   0174          if (PICMaster )
0053 1850          0175              BTFSC   DUMMY_PA, 0     ;
                   0176          else
                   0177              BTFSC   PORTA, 0        ;
                   0178          endif
0054 1412          0179              BSF     T2CON, 0        ;
                   0180 ;
                   0181          if (PICMaster )
0055 18D0          0182              BTFSC   DUMMY_PA, 1     ;
                   0183          else
                   0184              BTFSC   PORTA, 1        ;
```


© 1994 Microchip Technology Inc.

```
                        0185          endif
0056 1492               0186              BSF     T2CON, 1        ;
                        0187 ;
0057 170B               0188              BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0058 178B               0189              BSF     INTCON, GIE     ; Enable all Interrupts
0059 1410               0190              BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
005A 1512               0191              BSF     T2CON, TMR2ON   ; Turn Timer 2 ON
                        0192 ;
005B 285B               0193 lzz          goto    lzz             ; Loop waiting for TMR1 interrupt
                        0194 ;
                        0195 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                        0196 ;
005C 1E03               0197 OTHER_RESET   BTFSS  STATUS, TO      ; WDT Time-out?
005D 2807               0198 WDT_TIMEOUT   GOTO   ERROR1         ; YES, This is error condition
                        0199          if ( Debug_PU )
005E 2832               0200              goto   START           ; MCLR reset, Goto START
                        0201          else
                        0202              GOTO   MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                        0203          endif
                        0204 ;
                        0205          if (Debug )
005F 0000               0206 END_START     NOP                   ; END lable for debug
                        0207          endif
                        0208 ;
                        0209 ;
                        0210      org     PMEM_END               ; End of Program Memory
07FF 2807               0211              GOTO    ERROR1         ; If you get here your program was lost
                        0212
                        0213      end
                        0214
                        0215



MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ________ ________

0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ _______X

All other memory blocks unused.


Errors   :     0
Warnings :    17
```

© 1994 Microchip Technology Inc.

APPENDIX A2: PWM.H

```
        nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:        CLOCK.H
;       Revision:       7-13-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq                EQU             D'10000000'                     ; Device Frequency is 4 MHz
PULSE_TIME      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
;

DB_HI_BYTE              EQU             (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY EQU              (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR              EQU             40                                      ; RAM Location
OUTER_CNTR              EQU             41                                      ; RAM Location
;
T1OSO                   EQU             0                               ; The RC0 / T1OSO / T1CKI
;
RESET_V                 EQU             0x0000                  ; Address of RESET Vector
ISR_V                   EQU             0x0004                  ; Address of Interrupt Vector
PMEM_END                EQU             0x07FF                  ; Last address in Program Memory
TABLE_ADDR              EQU             0x0400                  ; Address where to start Tables
;
COUNTER         EQU     0x021           ;
;
XMIT_DATA       EQU     0x30
DATA_CNT        EQU     0x31
ONES_CNT        EQU     0x32
CCP1_INT_CNT    EQU     0x33
CCPREG_HI       EQU     0x40
CCPREG_LO       EQU     0x41
DUMMY_PA        EQU     0x50
DUMMY_PB        EQU     0x51
DUMMY_PC        EQU     0x52
DUMMY_PD        EQU     0x53
DUMMY_PE        EQU     0x54
DC_HI           EQU     0x55
DC_LO           EQU     0x56
T2_PERIOD       EQU     0xA0
;
        list
```


© 1994 Microchip Technology Inc.

```
MPASM 01.01 Released    COMP_1.ASM   7-13-1994  9:7:22                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program outputs a pulse train on the CCP1 pin, as specified by the
                     0006 ; values in the CCPR1H:CCPR1L.
                     0007 ;
                     0008 ;                    ____          ____      __
                     0009 ;  Pulse Train    ___|    |________|    |____|
                     0010 ;  Data Value          <-- 1 --> <- 0 ->
                     0011 ;                      ^         ^    ^
                     0012 ;                      |         |    |
                     0013 ;                      |  T_ONE_BIT  |  T_ZERO_BIT
                     0014 ;                      |             |
                     0015 ;                  PULSE_TIME    PULSE_TIME
                     0016 ;
                     0017 ;
                     0018 ;       Program = COMP_1.ASM
                     0019 ;       Revision Date:   7-13-94
                     0020 ;
                     0021 ;*****************************************************************************
                     0022 ;
                     0023 ;
                     0024 ; HARDWARE SETUP
                     0025 ;    PORTB         -  Data to serial transmit on CCP pin
                     0026 ;
                     0027 ;
                     0028             INCLUDE <C74_reg.h>
                     0247
                     0028
                     0029             INCLUDE <COMP.h>
                     0053
                     0029
                     0030 ;
0001                 0031 PICMaster       EQU     TRUE        ; A Debugging Flag
0001                 0032 Debug           EQU     TRUE        ; A Debugging Flag
0001                 0033 Debug_PU        EQU     TRUE        ; A Debugging Flag
                     0034 ;
                     0035 ;
                     0036 ; Reset address. Determine type of RESET
                     0037 ;
                     0038         org     RESET_V             ; RESET vector location
```

© 1994 Microchip Technology Inc.

```
0000 1683          0039 RESET       BSF     STATUS, RP0     ; Bank 1
0001 188E          0040             BTFSC   PCON, POR       ; Power-up reset?
0002 287C          0041             GOTO    START           ; YES
0003 28BA          0042             GOTO    OTHER_RESET     ; NO, a WDT or MCLR reset
                   0043 ;
                   0044 ; This is the Periperal Interrupt routine. Need to determine the type
                   0045 ; of interrupt that occurred. The following interrupts are enabled:
                   0046 ;   1.  CCP Capture Occured
                   0047 ;
                   0049         org     ISR_V               ; Interrupt vector location
                   0050 PER_INT_V
                   0051         if ( Debug )
0004 1405          0052             bsf     PORTA, 0        ; Turn on strobe
                   0053         endif                       ;
0005 1283          0054             BCF     STATUS, RP0     ; Bank 0
0006 190C          0055             BTFSC   PIR1, CCP1IF    ; Compare Interrupt occured?
0007 280E          0056             GOTO    CCP1_INT        ; YES, Service the TMR1 Interrupt
                   0057 ERROR1                              ; NO, Error Condition - Unknown Interrupt
0008 1505          0058             BSF     PORTA, 2        ; Toggle a PORT pin
0009 1105          0059             BCF     PORTA, 2
000A 2808          0060             GOTO    ERROR1
                   0061 ;
                   0062 ERROR2                              ; NO, Error Condition - Unknown Interrupt
000B 1585          0063             BSF     PORTA, 3        ; Toggle a PORT pin
000C 1185          0064             BCF     PORTA, 3
000D 280B          0065             GOTO    ERROR2
                   0066 ;
                   0067 ;
                   0068 ;*************************************************************************
                   0069 ; In the CCP interrupt.
                   0070 ; Since timer1 is not cleared on a CCP match, the value in the
                   0071 ; CCPR1H:CCPR1L register pair must be updated. This is done with
                   0072 ; a 16-bit add. Also after the 1st CCP1 match (CCP1 pin goes high)
                   0073 ; the next match will force it low. Depending on the value of the data bit
                   0074 ; determines the value add to the CCPR1H:CCPR1L register pair.
                   0075 ;
                   0076 ; After the data has been transmitted, the pin will have a sync pulse and
                   0077 ; then remain low for 300 us.
                   0078 ;*************************************************************************
                   0079 ;
                   0080
                   0081 CCP1_INT
000E 110C          0082             BCF     PIR1, CCP1IF    ; Clear CCP1 Interrupt Flag
000F 0AB3          0083             INCF    CCP1_INT_CNT    ;
0010 1C33          0084             BTFSS   CCP1_INT_CNT, 0 ;
0011 2833          0085             GOTO    SYNC_PULSE
                   0086 DATA_PULSE
0012 03B1          0087             DECF    DATA_CNT        ; Decrement the Count of data bits
```


© 1994 Microchip Technology Inc.

```
0013 1903          0088                BTFSC   STATUS, Z           ; Have we transmitted all the Data Bits?
0014 2827          0089                GOTO    PERIOD_DELTA        ; YES, Delay to 300 us
0015 0DB0          0090                RLF     XMIT_DATA, F        ; NO, get next bit to transmit
0016 1803          0091                BTFSC   STATUS, C           ; Is the bit to transmit a '1'?
0017 281F          0092                GOTO    ONE_DATA            ; YES, Stay low for 17.6 us
                   0093 ZERO_DATA
0018 302F          0094                MOVLW   LOW ( T_ZERO_BIT )  ; NO, Stay low for 8.8 us
0019 0795          0095                ADDWF   CCPR1L, F           ; Update Compare register pair latch
001A 1803          0096                BTFSC   STATUS, C           ;
001B 0A96          0097                INCF    CCPR1H, F           ;
001C 3000          0098                MOVLW   HIGH (T_ZERO_BIT )  ;
001D 0796          0099                ADDWF   CCPR1H, F           ;
001E 287A          0100                GOTO    RET_FIE
                   0101 ;
                   0102 ONE_DATA
001F 305E          0103                MOVLW   LOW ( T_ONE_BIT )   ; Stay low for 17.6 us
0020 0795          0104                ADDWF   CCPR1L, F           ; Update Compare register pair latch
0021 1803          0105                BTFSC   STATUS, C           ;
0022 0A96          0106                INCF    CCPR1H, F           ;
0023 3000          0107                MOVLW   HIGH ( T_ONE_BIT )  ;
0024 0796          0108                ADDWF   CCPR1H, F           ;
0025 0AB2          0109                INCF    ONES_CNT            ; Increment the number of '1's in the byte
0026 287A          0110                GOTO    RET_FIE
                   0111 ;
                   0112 PERIOD_DELTA
0027 0832          0113                MOVF    ONES_CNT, W         ;
0028 390F          0114                ANDLW   0x0F                ; Only want 9 states (0 1s to 8 1s)
0029 0782          0115                ADDWF   PCL, F              ;
002A 283B          0116                GOTO    ZERO_1              ; There was 0 ones in the data byte
002B 2842          0117                GOTO    ONE_1               ; There was 1 one in the data byte
002C 2849          0118                GOTO    TWO_1               ; There was 2 ones in the data byte
002D 2850          0119                GOTO    THREE_1             ; There was 3 ones in the data byte
002E 2857          0120                GOTO    FOUR_1              ; There was 4 ones in the data byte
002F 285E          0121                GOTO    FIVE_1              ; There was 5 ones in the data byte
0030 2865          0122                GOTO    SIX_1               ; There was 6 ones in the data byte
0031 286C          0123                GOTO    SEVEN_1             ; There was 7 ones in the data byte
0032 2873          0124                GOTO    EIGHT_1             ; There was 8 ones in the data byte
                   0125 ;
                   0126 SYNC_PULSE
0033 302F          0127                MOVLW   LOW ( PULSE_TIME )  ; Update Compare register pair latch
0034 0795          0128                ADDWF   CCPR1L, F           ;
0035 1803          0129                BTFSC   STATUS, C           ;
0036 0A96          0130                INCF    CCPR1H, F           ;
0037 3000          0131                MOVLW   HIGH ( PULSE_TIME ) ;
0038 0796          0132                ADDWF   CCPR1H, F           ;
0039 1417          0133                BSF     CCP1CON, 0          ; On Compare match, CCP1 pin = L
003A 0009          0134                RETFIE
                   0135 ;
```


© 1994 Microchip Technology Inc.

```
                0136 ZERO_1
003B 30EC       0137            MOVLW   LOW ( ZERO_1S )   ; Update Compare register pair latch
003C 0795       0138            ADDWF   CCPR1L, F         ;
003D 1803       0139            BTFSC   STATUS, C         ;
003E 0A96       0140            INCF    CCPR1H, F         ;
003F 3002       0141            MOVLW   HIGH ( ZERO_1S )  ;
0040 0796       0142            ADDWF   CCPR1H, F         ;
0041 287A       0143            GOTO    RET_FIE
                0144 ;
                0145 ONE_1
0042 30BD       0146            MOVLW   LOW ( ONE_1S )    ; Update Compare register pair latch
0043 0795       0147            ADDWF   CCPR1L, F         ;
0044 1803       0148            BTFSC   STATUS, C         ;
0045 0A96       0149            INCF    CCPR1H, F         ;
0046 3002       0150            MOVLW   HIGH ( ONE_1S )   ;
0047 0796       0151            ADDWF   CCPR1H, F         ;
0048 287A       0152            GOTO    RET_FIE
                0153 ;
                0154 TWO_1
0049 308E       0155            MOVLW   LOW ( TWO_1S )    ; Update Compare register pair latch
004A 0795       0156            ADDWF   CCPR1L, F         ;
004B 1803       0157            BTFSC   STATUS, C         ;
004C 0A96       0158            INCF    CCPR1H, F         ;
004D 3002       0159            MOVLW   HIGH ( TWO_1S )   ;
004E 0796       0160            ADDWF   CCPR1H, F         ;
004F 287A       0161            GOTO    RET_FIE
                0162 ;
                0163 THREE_1
0050 305F       0164            MOVLW   LOW ( THREE_1S )  ; Update Compare register pair latch
0051 0795       0165            ADDWF   CCPR1L, F         ;
0052 1803       0166            BTFSC   STATUS, C         ;
0053 0A96       0167            INCF    CCPR1H, F         ;
0054 3002       0168            MOVLW   HIGH ( THREE_1S ) ;
0055 0796       0169            ADDWF   CCPR1H, F         ;
0056 287A       0170            GOTO    RET_FIE
                0171 ;
                0172 FOUR_1
0057 3030       0173            MOVLW   LOW ( FOUR_1S )   ; Update Compare register pair latch
0058 0795       0174            ADDWF   CCPR1L, F         ;
0059 1803       0175            BTFSC   STATUS, C         ;
005A 0A96       0176            INCF    CCPR1H, F         ;
005B 3002       0177            MOVLW   HIGH ( FOUR_1S )  ;
005C 0796       0178            ADDWF   CCPR1H, F         ;
005D 287A       0179            GOTO    RET_FIE
                0180 ;
                0181 FIVE_1
005E 3001       0182            MOVLW   LOW ( FIVE_1S )   ; Update Compare register pair latch
005F 0795       0183            ADDWF   CCPR1L, F         ;
```


© 1994 Microchip Technology Inc.

```
0060 1803          0184            BTFSC   STATUS, C           ;
0061 0A96          0185            INCF    CCPR1H, F           ;
0062 3002          0186            MOVLW   HIGH ( FIVE_1S )    ;
0063 0796          0187            ADDWF   CCPR1H, F           ;
0064 287A          0188            GOTO    RET_FIE
                   0189 ;
                   0190 SIX_1
0065 30D2          0191            MOVLW   LOW ( SIX_1S )      ; Update Compare register pair latch
0066 0795          0192            ADDWF   CCPR1L, F           ;
0067 1803          0193            BTFSC   STATUS, C           ;
0068 0A96          0194            INCF    CCPR1H, F           ;
0069 3001          0195            MOVLW   HIGH ( SIX_1S )     ;
006A 0796          0196            ADDWF   CCPR1H, F           ;
006B 287A          0197            GOTO    RET_FIE
                   0198 ;
                   0199 SEVEN_1
006C 30A3          0200            MOVLW   LOW ( SEVEN_1S )    ; Update Compare register pair latch
006D 0795          0201            ADDWF   CCPR1L, F           ;
006E 1803          0202            BTFSC   STATUS, C           ;
006F 0A96          0203            INCF    CCPR1H, F           ;
0070 3001          0204            MOVLW   HIGH ( SEVEN_1S )   ;
0071 0796          0205            ADDWF   CCPR1H, F           ;
0072 287A          0206            GOTO    RET_FIE
                   0207 ;
                   0208 EIGHT_1
0073 3074          0209            MOVLW   LOW ( EIGHT_1S )    ; Update Compare register pair latch
0074 0795          0210            ADDWF   CCPR1L, F           ;
0075 1803          0211            BTFSC   STATUS, C           ;
0076 0A96          0212            INCF    CCPR1H, F           ;
0077 3001          0213            MOVLW   HIGH ( EIGHT_1S )   ;
0078 0796          0214            ADDWF   CCPR1H, F           ;
0079 287A          0215            GOTO    RET_FIE
                   0216
                   0217 RET_FIE
007A 1017          0218            BCF     CCP1CON, 0          ; On Compare match, CCP1 pin = H
007B 0009          0219            RETFIE                      ; Return / Enable Global Interrupts
                   0220 ;
                   0222 ;
                   0223 ;*******************************************************************************
                   0224 ;*****      Start program here, Power-On Reset occurred.
                   0225 ;*******************************************************************************
                   0226 ;
                   0227 START                                  ; POWER_ON Reset (Beginning of program)
007C 1283          0228            BCF     STATUS, RP0         ; Bank 0
007D 018F          0229            CLRF    TMR1H               ;
007E 018E          0230            CLRF    TMR1L               ;
                   0231 ;
                   0232 MCLR_RESET                             ; A Master Clear Reset
```

```
007F 1283          0233            BCF     STATUS, RP0     ; Bank 0
0080 0183          0234            CLRF    STATUS          ; Do initialization (Bank 0)
0081 018B          0235            CLRF    INTCON
0082 018C          0236            CLRF    PIR1
0083 1683          0237            BSF     STATUS, RP0     ; Bank 1
0084 3080          0238            MOVLW   0x80            ; Disable PORTB weak pull-ups
0085 0081          0239            MOVWF   OPTION_R        ;
0086 018C          0240            CLRF    PIE1            ; Disable all peripheral interrupts
0087 30FF          0241            MOVLW   0xFF            ;
0088 009F          0242            MOVWF   ADCON1          ; Port A is Digital.
                   0243 ;**********************************************************************
                   0244 ;
0089 1283          0245            BCF     STATUS, RP0     ; Bank 0
008A 0185          0246            CLRF    PORTA           ; ALL PORT output should output Low.
008B 0186          0247            CLRF    PORTB
008C 0187          0248            CLRF    PORTC
008D 0188          0249            CLRF    PORTD
008E 0189          0250            CLRF    PORTE
008F 1010          0251            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                   0252 ;
0090 1683          0253            BSF     STATUS, RP0     ; Select Bank 1
0091 0185          0254            CLRF    TRISA           ; RA5 -  0 outputs
0092 30FF          0255            MOVLW   0xFF            ;
0093 0086          0256            MOVWF   TRISB           ; RB Port are inputs
0094 0187          0257            CLRF    TRISC           ; RC Port are outputs
0095 0188          0258            CLRF    TRISD           ; RD Port are outputs
0096 0189          0259            CLRF    TRISE           ; RE Port are outputs
0097 150C          0260            BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
0098 1283          0261            BCF     STATUS, RP0     ; Select Bank 0
                   0262 ;**********************************************************************
                   0264 ;
                   0265 ;
                   0266 ; Initialize the Special Function Registers (SFR) interrupts
                   0267 ;
0099 018C          0268            CLRF    PIR1            ;
009A 0190          0269            CLRF    T1CON           ; Timer mode
009B 170B          0270            BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
009C 178B          0271            BSF     INTCON, GIE     ; Enable all Interrupts
                   0272 ;**********************************************************************
                   0273 ; Set-up timer and compare latches and then turn timer1 on.
                   0274 ;
009D 1010          0275            BCF     T1CON, TMR1ON   ; Turn OFF timer1
009E 3041          0276            MOVLW   CCPREG_HI       ; TMR1 = CCPR1H:CCPR1L - 1
009F 008F          0277            MOVWF   TMR1H           ;
00A0 3042          0278            MOVLW   CCPREG_LO       ;
00A1 008E          0279            MOVWF   TMR1L           ;
00A2 038E          0280            DECF    TMR1L           ;
00A3 1803          0281            BTFSC   STATUS, C       ;
```


© 1994 Microchip Technology Inc.

```
00A4 038F            0282              DECF    TMR1H           ;
00A5 3008            0283              MOVLW   0x08            ; On match CCP1 = H level
00A6 0097            0284              MOVWF   CCP1CON         ;
00A7 3009            0285              MOVLW   0x09            ;
00A8 00B1            0286              MOVWF   DATA_CNT        ; 8-bits to transfer
00A9 01B2            0287              CLRF    ONES_CNT        ; Result after xmit holds the number of 1's in a byte
00AA 30FF            0288              MOVLW   0xFF            ;
00AB 00B3            0289              MOVWF   CCP1_INT_CNT    ; No CCP1 transmit interrups yet
00AC 1410            0290              BSF     T1CON, TMR1ON   ; Turn ON timer1
                     0291 ;
                     0292 ;
                     0293 ; This code segment is an infinite loop that will always transmit the data
                     0294 ; contained in the XMIT_DATA register. After each byte is transmitted a new
                     0295 ; byte is read. If using PICMASTER (in stand alone mode), this is read from
                     0296 ; a register that is updated after a break (at NOP). If in a system, PORTB
                     0297 ; is read. All other variables are reinitalized after each byte.
                     0298 ;
                     0299 NEXT_BYTE
                     0300 ;
00AD 0831            0301 WAIT            MOVF    DATA_CNT, w     ;
00AE 1D03            0302                 BTFSS   STATUS, Z       ; Is DATA_CNT = 0 ?
00AF 28AD            0303                 GOTO    WAIT            ; NO, must wait until YES
00B0 0000            0304                 NOP                     ;
                     0305             if ( Debug )
00B1 1005            0306                 bcf     PORTA, 0        ; Turn off strobe
                     0307             endif                       ;
                     0308
                     0309             if ( PICMaster )
00B2 0840            0310                 MOVF    DUMMY_PB, W     ;
                     0311             else
                     0312                 MOVF    PORTB, W        ;
                     0313             endif
00B3 00B0            0314                 MOVWF   XMIT_DATA       ; New data to transmit
00B4 30FF            0315                 MOVLW   0xFF            ;
00B5 00B3            0316                 MOVWF   CCP1_INT_CNT    ;
00B6 3009            0317                 MOVLW   0x09            ;
00B7 00B1            0318                 MOVWF   DATA_CNT        ;
00B8 01B2            0319                 CLRF    ONES_CNT        ;
00B9 28AD            0320                 GOTO    NEXT_BYTE       ;
                     0321 ;
                     0322 ;
                     0323 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                     0324 ;
00BA 1E03            0325 OTHER_RESET     BTFSS   STATUS, TO      ; WDT Time-out?
00BB 2808            0326 WDT_TIMEOUT     GOTO    ERROR1          ; YES, This is error condition
                     0327             if ( Debug_PU )
00BC 287C            0328                 goto    START           ; MCLR reset, Goto START
                     0329             else
```


© 1994 Microchip Technology Inc.

```
                        0330            GOTO    MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                        0331        endif
                        0332 ;
                        0333        if (Debug )
00BD 0000               0334 END_START    NOP                   ; END lable for debug
                        0335        endif
                        0336 ;
                        0337 ;
                        0338    org     PMEM_END                ; End of Program Memory
07FF 2808               0339            GOTO    ERROR1          ; If you get here your program was lost
                        0340
                        0341    end
                        0342
                        0343
                        0344


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-
00C0 : ________ ________ ________ ________
0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ _______X

All other memory blocks unused.


Errors   :     0
Warnings :    10
```


© 1994 Microchip Technology Inc.

```
        nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:        CLOCK.H
;       Revision:       7-13-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq        EQU     D'10000000'                     ; Device Frequency is 4 MHz
PULSE_TIME      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ZERO_BIT      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ONE_BIT       EQU     ( Dev_Freq / D'4000' ) * D'376' / D'10000' )
;
ZERO_1S         EQU     ((( Dev_Freq / D'4000' ) * (D'6000' - (D'16' * D'188'))) / D'10000' )
ONE_1S          EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (3 * D'188' + D'14' * D'188')) / D'10000')
TWO_1S          EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (6 * D'188' + D'12' * D'188')) / D'10000')
THREE_1S        EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'9' * D'188' + D'10' * D'188')) / D'10000')
FOUR_1S         EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'12' * D'188' + 8 * D'188')) / D'10000')
FIVE_1S         EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'15' * D'188' + 6 * D'188')) / D'10000')
SIX_1S          EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'18' * D'188' + 4 * D'188')) / D'10000')
SEVEN_1S        EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'21' * D'188' + 2 * D'188')) / D'10000')
EIGHT_1S        EQU     ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'24' * D'188') ) / D'10000' )

DB_HI_BYTE      EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY  EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR      EQU     40                              ; RAM Location
OUTER_CNTR      EQU     41                              ; RAM Location
;
T1OSO           EQU     0                               ; The RC0 / T1OSO / T1CKI
;
RESET_V         EQU     0x0000                          ; Address of RESET Vector
ISR_V           EQU     0x0004                          ; Address of Interrupt Vector
PMEM_END        EQU     0x07FF                          ; Last address in Program Memory
TABLE_ADDR      EQU     0x0400                          ; Address where to start Tables
;
COUNTER         EQU     0x021                           ;
;
XMIT_DATA       EQU     0x30
DATA_CNT        EQU     0x31
ONES_CNT        EQU     0x32
CCP1_INT_CNT    EQU     0x33
DUMMY_PB        EQU     0x40
CCPREG_HI       EQU     0x41
CCPREG_LO       EQU     0x42
;
        list
```


© 1994 Microchip Technology Inc.

```
MPASM 01.01 Released    CAPT_2.ASM   7-19-1994  10:54:15              PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program implements a real time clock using the TMR1 module of the
                     0006 ; PIC16Cxx family.
                     0007 ;
                     0008 ;       Program = CAPT_2.ASM
                     0009 ;       Revision Date:   7-19-94
                     0010 ;
                     0011 ;*****************************************************************************
                     0012 ;
                     0013 ;
                     0014 ; HARDWARE SETUP
                     0015 ;
                     0016 ;           CCP2 Compare Output
                     0017 ;           CCP1 Capture Input
                     0018 ;           CCP2  ->  CCP1
                     0019 ;
                     0020 ;
                     0021             INCLUDE <C74_reg.h>
                     0249
                     0021
                     0022             INCLUDE <CAPT.h>
                     0022
                     0023 ;
                     0024 ;
0001                 0025 PICMaster       EQU     TRUE            ; A Debugging Flag
0001                 0026 Debug           EQU     TRUE            ; A Debugging Flag
0001                 0027 Debug_PU        EQU     TRUE            ; A Debugging Flag
                     0028 ;
                     0029 ;
                     0030 ; Reset address. Determine type of RESET
                     0031 ;
                     0032         org     RESET_V                 ; RESET vector location
0000 1683            0033 RESET       BSF     STATUS, RP0         ; Bank 1
0001 188E            0034             BTFSC   PCON, POR           ; Power-up reset?
0002 282F            0035             GOTO    START               ; YES
0003 2861            0036             GOTO    OTHER_RESET         ; NO, a WDT or MCLR reset
                     0037 ;
                     0038 ; This is the Periperal Interrupt routine. Need to determine the type
                     0039 ; of interrupt that occurred. The following interrupts are enabled:
```


© 1994 Microchip Technology Inc.

```
                        0040 ;   1.  CCP1 Capture Occured
                        0041 ;   2.  CCP2 Compare Occured
                        0042 ;
                        0044         org     ISR_V                  ; Interrupt vector location
                        0045 PER_INT_V
0004 1283               0046             BCF     STATUS, RP0        ; Bank 0
0005 190C               0047             BTFSC   PIR1, CCP1IF       ; CCP1 Interrupt occured? (Capture)
0006 281D               0048             GOTO    CAPTURE            ; YES, Service the CCP1 Interrupt
0007 180D               0049             BTFSC   PIR2, CCP2IF       ; CCP2 Interrupt occured? (Compare)
0008 2811               0050             GOTO    COMPARE            ; YES, Service the CCP2 Interrupt
0009 180C               0051             BTFSC   PIR1, TMR1IF       ; NO, Timer 1 Overflow?
000A 282D               0052             GOTO    T1OVFL             ; YES,
                        0053 ERROR1                                 ; NO, Error Condition - Unknown Interrupt
000B 1488               0054             BSF     PORTD, 1           ; Toggle a PORT pin
000C 1088               0055             BCF     PORTD, 1
000D 280B               0056             GOTO    ERROR1
                        0057 ;
                        0058 ERROR2                                 ; NO, Error Condition - Unknown Interrupt
000E 1508               0059             BSF     PORTD, 2           ; Toggle a PORT pin
000F 1108               0060             BCF     PORTD, 2
0010 280E               0061             GOTO    ERROR2
                        0062 ;
                        0063 ; The Compare generates a Square wave based on the value on PORTB (in DUMMY_PB)
                        0064 ; and on PORTD (in DUMMY_PD). PORTB is loaded into low compare latch and PORTD
                        0065 ; is loaded into the high compare latch. If the value of the ports is not changed,
                        0066 ; a capture overflow condition will occur when PORTD:PORTB > 7Fh. This overflow
                        0067 ; is only indicated by the time between captures being much less then expected.
                        0068 ;
                        0069 COMPARE
0011 100D               0070             BCF     PIR2, CCP2IF       ; Clear CCP2 Interrupt Flag
                        0071         if ( PICMaster )
0012 0851               0072             MOVF    DUMMY_PB, W        ;
                        0073         else
                        0074             MOVF    PORTB, W           ;
                        0075         endif
0013 079B               0076             ADDWF   CCPR2L, F          ; Update Compare register pair latch
0014 1803               0077             BTFSC   STATUS, C          ;
0015 0A9C               0078             INCF    CCPR2H, F          ;
                        0079         if ( PICMaster )
0016 0853               0080             MOVF    DUMMY_PD, W        ;
                        0081         else
                        0082             MOVF    PORTD, W           ;
                        0083         endif
0017 079C               0084             ADDWF   CCPR2H, F          ;
0018 0AB3               0085             INCF    CCP2_INT_CNT       ;
0019 141D               0086             BSF     CCP2CON, 0         ; On Compare match, CCP2 pin = L
001A 1C33               0087             BTFSS   CCP2_INT_CNT, 0    ;
001B 101D               0088             BCF     CCP2CON, 0         ; On Compare match, CCP2 pin = H
```

© 1994 Microchip Technology Inc.

```
                    0089 END_COMPARE
001C 0009           0090            RETFIE          ; Return / Enable Global Interrupts
                    0091
                    0092 ;
                    0093 ; The result of the new capture minus the old capture is stored in the new capture
                    0094 ; registers (CAPT_NEW_H:CAPT_NEW_L)
                    0095 ;
                    0096 CAPTURE
001D 110C           0097            BCF     PIR1, CCP1IF    ; Clear CCP1 Interrupt Flag
001E 0815           0098            MOVF    CCPR1L, W       ; New capture value (low byte)
001F 00C1           0099            MOVWF   CAPT_NEW_L      ;
0020 0816           0100            MOVF    CCPR1H, W       ; New capture value (high byte)
0021 00C0           0101            MOVWF   CAPT_NEW_H      ;
                    0102 ;
0022 0843           0103            MOVF    CAPT_OLD_L, W   ;
0023 02C1           0104            SUBWF   CAPT_NEW_L, F   ; Subtract the low bytes of the 2 captures
0024 1C03           0105            BTFSS   STATUS, C       ; Did a borrow occur?
0025 03C0           0106            DECF    CAPT_NEW_H, F   ; YES, Decrement old capture (high byte)
0026 0842           0107            MOVF    CAPT_OLD_H, W   ; New capture value (low byte)
0027 02C0           0108            SUBWF   CAPT_NEW_H, F   ; Subtract the low bytes of the 2 captures
0028 0815           0109 LOAD_OLD   MOVF    CCPR1L, W       ; New capture value (low byte)
0029 00C3           0110            MOVWF   CAPT_OLD_L      ;
002A 0816           0111            MOVF    CCPR1H, W       ; New capture value (high byte)
002B 00C2           0112            MOVWF   CAPT_OLD_H      ;
                    0113 END_CAPTURE
002C 0009           0114            RETFIE
                    0115 ;
                    0116 ;
                    0117 T1OVFL
002D 100C           0118            BCF     PIR1, TMR1IF    ; Clear T1 Overflow Interrupt Flag
002E 0009           0119            RETFIE                  ; Return / Enable Global Interrupts
                    0120 ;
                    0121 ;
                    0122 ;*******************************************************************************
                    0123 ;*****      Start program here, Power-On Reset occurred.
                    0124 ;*******************************************************************************
                    0125 ;
                    0126 START                              ; POWER_ON Reset (Beginning of program)
002F 1283           0127            BCF     STATUS, RP0     ; Bank 0
0030 018F           0128            CLRF    TMR1H           ;
0031 018E           0129            CLRF    TMR1L           ;
                    0130 ;
                    0131 MCLR_RESET                         ; A Master Clear Reset
0032 1283           0132            BCF     STATUS, RP0     ; Bank 0
0033 0183           0133            CLRF    STATUS          ; Do initialization (Bank 0)
0034 018B           0134            CLRF    INTCON
0035 018C           0135            CLRF    PIR1
0036 1683           0136            BSF     STATUS, RP0     ; Bank 1
```


© 1994 Microchip Technology Inc.

```
0037 3000           0137            MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
0038 0081           0138            MOVWF   OPTION_R        ;
0039 018C           0139            CLRF    PIE1            ; Disable all peripheral interrupts
003A 018D           0140            CLRF    PIE2            ; Disable all peripheral interrupts
003B 30FF           0141            MOVLW   0xFF            ;
003C 009F           0142            MOVWF   ADCON1          ; Port A is Digital.
                    0143 ;
                    0144 ;
003D 1283           0145            BCF     STATUS, RP0     ; Bank 0
003E 0185           0146            CLRF    PORTA           ; ALL PORT output should output Low.
003F 0186           0147            CLRF    PORTB
0040 0187           0148            CLRF    PORTC
0041 0188           0149            CLRF    PORTD
0042 0189           0150            CLRF    PORTE
0043 1010           0151            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                    0152 ;
0044 1683           0153            BSF     STATUS, RP0     ; Select Bank 1
0045 0185           0154            CLRF    TRISA           ; RA5 -  0 outputs
0046 30FF           0155            MOVLW   0xFF            ;
0047 0086           0156            MOVWF   TRISB           ; RB7 - 0 inputs
0048 0187           0157            CLRF    TRISC           ; RC Port are outputs
0049 1507           0158            BSF     TRISC, 2        ; CCP1 is an INPUT
004A 0088           0159            MOVWF   TRISD           ; RD Port are inputs
004B 0189           0160            CLRF    TRISE           ; RE Port are outputs
004C 150C           0161            BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
004D 140D           0162            BSF     PIE2, CCP2IE    ; Enable CCP2 Interrupt
004E 1283           0163            BCF     STATUS, RP0     ; Select Bank 0
                    0164 ;
                    0165 ;
                    0166 ; Initialize the Special Function Registers (SFR) interrupts
                    0167 ;
004F 018C           0168            CLRF    PIR1            ;
0050 018D           0169            CLRF    PIR2            ;
0051 0190           0170            CLRF    T1CON           ; Timer mode
0052 170B           0171            BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0053 178B           0172            BSF     INTCON, GIE     ; Enable all Interrupts
                    0173 ;
                    0174 ; Set-up timer and compare latches and then turn timer1 on.
                    0175 ;
0054 1010           0176            BCF     T1CON, TMR1ON   ; Turn OFF timer1
                    0177        if ( PICMaster )
0055 0851           0178            MOVF    DUMMY_PB, W     ;
                    0179        else
                    0180            MOVF    PORTB, W        ;
                    0181        endif
0056 079B           0182            ADDWF   CCPR2L, F       ; Update Compare register pair latch
0057 1803           0183            BTFSC   STATUS, C       ;
0058 0A9C           0184            INCF    CCPR2H, F       ;
```

```
                        0185          if ( PICMaster )
0059 08D3               0186              MOVF    DUMMY_PD          ;
                        0187          else
                        0188              MOVF    PORTD, W          ;
                        0189          endif
005A 079C               0190              ADDWF   CCPR2H, F         ;
005B 3008               0191              MOVLW   0x08              ; On match CCP2 = H level
005C 009D               0192              MOVWF   CCP2CON           ;
005D 3005               0193              MOVLW   0x05              ; Capture on every rising edge
005E 0097               0194              MOVWF   CCP1CON           ;
005F 1410               0195              BSF     T1CON, TMR1ON     ; Turn ON timer1
                        0196 ;
                        0197 ;
                        0198 ;
0060 2860               0199 lzz          goto    lzz               ; Loop waiting for interrupts (for use with PICMASTER)
                        0200 ;
                        0201 ;
                        0202 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                        0203 ;
0061 1E03               0204 OTHER_RESET    BTFSS   STATUS, TO      ; WDT Time-out?
0062 280B               0205 WDT_TIMEOUT    GOTO    ERROR1          ; YES, This is error condition
                        0206          if ( Debug_PU )
0063 282F               0207              goto    START           ; MCLR reset, Goto START
                        0208          else
                        0209              GOTO    MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                        0210          endif
                        0211 ;
                        0212          if (Debug )
0064 0000               0213 END_START      NOP                     ; END lable for debug
                        0214          endif
                        0215 ;
                        0216 ;
                        0217      org     PMEM_END                  ; End of Program Memory
07FF 280B               0218              GOTO    ERROR1            ; If you get here your program was lost
                        0219
                        0220      end
                        0221
                        0222
```


© 1994 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX---------- ----------

0780 : ---------- ---------- ---------- ----------
07C0 : ---------- ---------- ---------- ----------X

All other memory blocks unused.


Errors   :     0
Warnings :    13
```

© 1994 Microchip Technology Inc.

```
        nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:        CLOCK.H
;       Revision:       7-19-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq                EQU             D'4000000'              ; Device Frequency is 4 MHz
DB_HI_BYTE              EQU             (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY          EQU             (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR              EQU             40                      ; RAM Location
OUTER_CNTR              EQU             41                      ; RAM Location
;
T1OSO                   EQU             0                       ; The RC0 / T1OSO / T1CKI
;
RESET_V                 EQU             0x0000                  ; Address of RESET Vector
ISR_V                   EQU             0x0004                  ; Address of Interrupt Vector
PMEM_END                EQU             0x07FF                  ; Last address in Program Memory
TABLE_ADDR              EQU             0x0400                  ; Address where to start Tables
;
COUNTER                 EQU             0x021                   ;
CCP2_INT_CNT            EQU             0x33
;
;
; DUMMY_PD:DUMMY_PB contain the value to be loaded into the CCP2 compare registers
; (CCPR2H:CCPR2L)
;
DUMMY_PA                EQU             0x50
DUMMY_PB                EQU             0x51
DUMMY_PC                EQU             0x52
DUMMY_PD                EQU             0x53
DUMMY_PE                EQU             0x54
;
;
; CAPT_NEW_H:CAPT_NEW_L stores the NEW captured value and the result of the
; subtraction between this capture and the previous.
;    CAPT_NEW_H:CAPT_NEW_L  =  CAPT_NEW_H:CAPT_NEW_L - CAPT_OLD_H:CAPT_OLD_L
;
; After all computations the new capture value is moved to the CAPT_OLD_H:CAPT_OLD_L
; in preperation for the next capture value.
;
CAPT_NEW_H              EQU             0x040                   ;
CAPT_NEW_L              EQU             0x041                   ;
CAPT_OLD_H              EQU             0x042                   ;
CAPT_OLD_L              EQU             0x043                   ;
;
        list
```


© 1994 Microchip Technology Inc.

Interfacing to an LCD Module

INTRODUCTION

This application note interfaces a PIC16CXX device to the Hitachi LM032L LCD character display module. This module is a two line by twenty character display. LCD modules are useful for displaying text information from a system. In large volume applications, the use of custom LCD displays become economical. These routines should be a good starting point for users whose application implement a custom LCD. This source code should be compatible with the PIC16C5X devices, after modifications for the special function register initialization, but has not been verified on those devices.

OPERATION

The Hitachi® LM032L LCD character display module can operate in one of two modes. The first (and default) mode is the 4-bit data interface mode. The second is the 8-bit data interface mode. When operating in 4-bit mode, two transfers per character / command are required. 8-bit mode, though easier to implement (less program memory) requires four additional I/O lines. The use of 8-bit mode is strictly a program memory size / I/O trade-off. The three most common data interfaces from the microcontroller are:

1. An 8-bit interface.
2. A 4-bit interface, with data transfers on the high nibble of the port.
3. A 4-bit interface, with data transfers on the low nibble of the port.

The LCD module also has three control Signal, Enable (E), Read / Write (R_W), and Register Select (RS). These functions of these control signals are show in Table 1.

TABLE 1: CONTROL SIGNAL FUNCTIONS

Control Signal	Function
E	Causes data / control state to be latched Rising Edge = Latches control state (RS and R_W) Falling Edge = Latches data
RS	Register Select Control 0 = LCD in command mode 1 = LCD in data mode
R_W	Read / Write control 0 = LCD to read data 1 = LCD to write data

A single source file, with conditional assemble is used to generate these three options. This requires two flags. The flags and their results are shown in Table 2.

TABLE 2: CONDITIONAL ASSEMBLY FLAGS

Flags		
Four_bit	Data_HI	Result
1	0	4-bit mode. Data transferred on the low nibble of the port.
1	1	4-bit mode. Data transferred on the high nibble of the port.
0	X	8-bit mode.

© 1994 Microchip Technology Inc.

Figure 1A through Figure 1C show the block diagrams for the three different data interfaces. The LCD_CNTL and LCD_DATA lines are user definable to their port assignment. This is accomplished with EQUate statements in the source code. See Appendices B - D.

FIGURE 1A: 8-BIT DATA INTERFACE

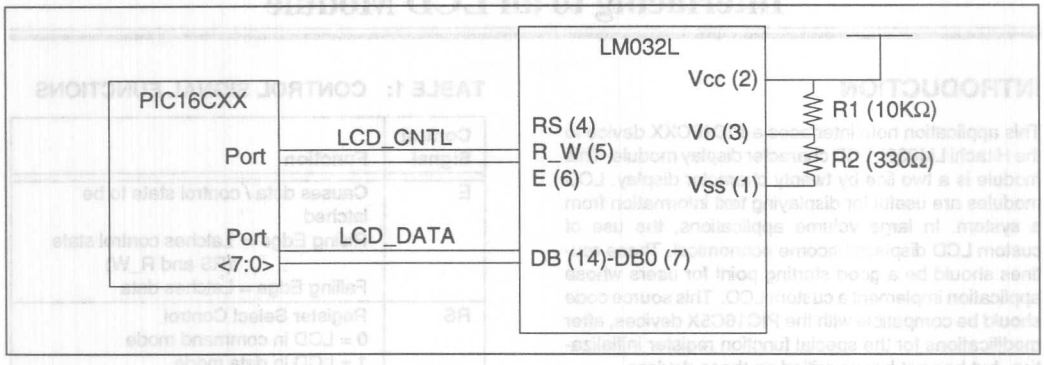


FIGURE 1B: 4-BIT MODE. DATA TRANSFERRED ON THE HIGH NIBBLE OF THE PORT

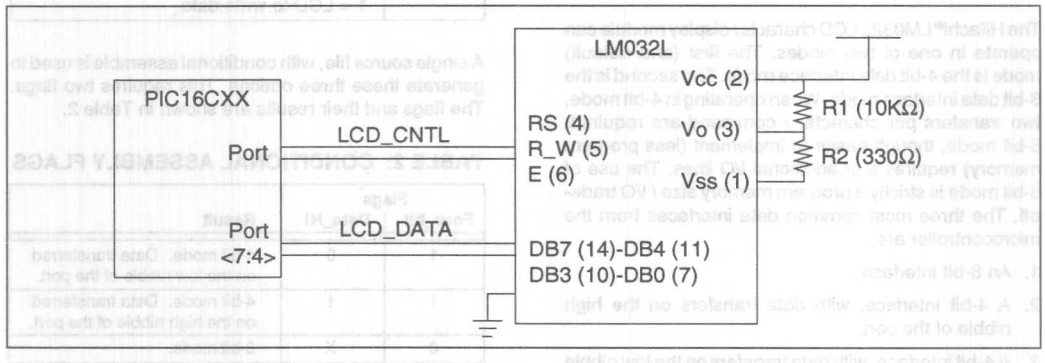
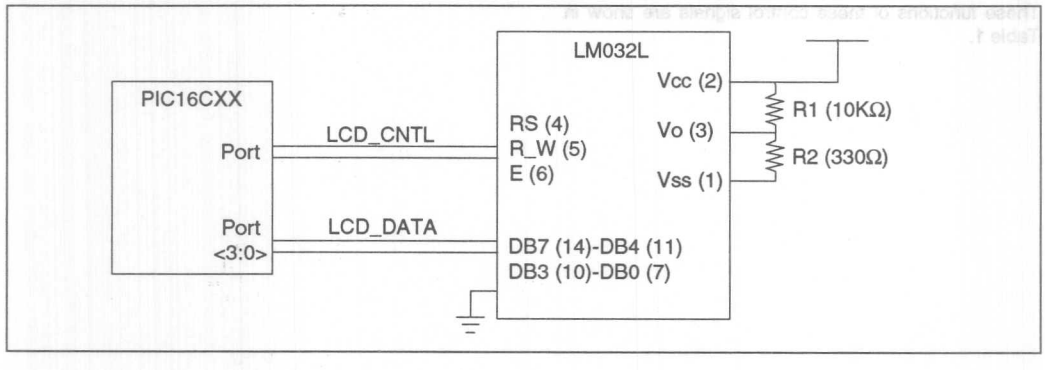


FIGURE 1C: 4-BIT MODE. DATA TRANSFERRED ON THE LOW NIBBLE OF THE PORT



© 1994 Microchip Technology Inc.

LCD 's (drivers) are slow devices when compared to microcontrollers. Care must be taken from having communication occur to quickly. The timing requirements of the LM032L are shown in Appendix A. It is recommended that the complete specifications of the LM032L be acquired from Hitachi or an Hitachi distributor. The literature number is CE-E613Q and M24T013 for the display driver.

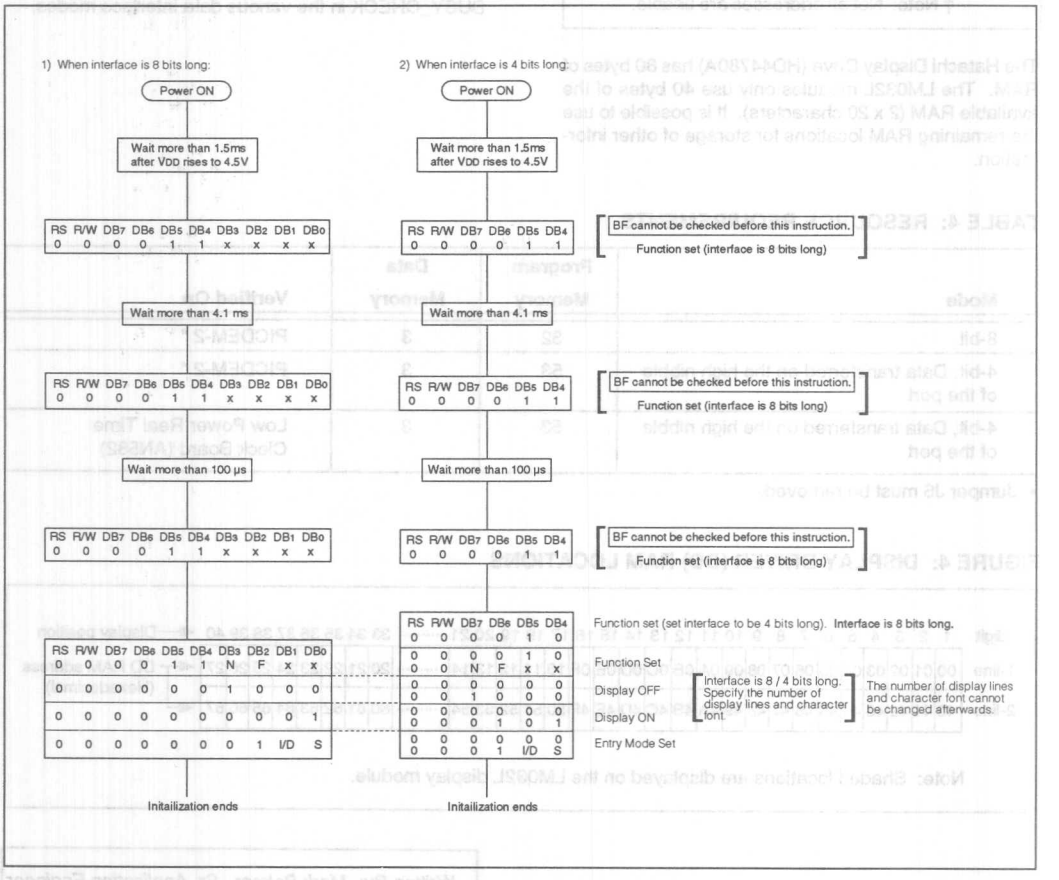
When the module powers up, the default data transfer mode is 8-bit. The initialization sequence only requires commands that are 4-bit in length. The last initialization command then needs to be sent to the display to specify the data transfer width (4- or 8-bit). Then a delay of 4.6 ms must be executed before the LCD module can be initialized. Some of the LCD module commands are:

- 1 or 2 lines of characters
- Display on /off
- Clear display
- Increment / do not increment character address pointer after each character
- Load character address pointer

The initialization flow for the module is shown in Figure 2.

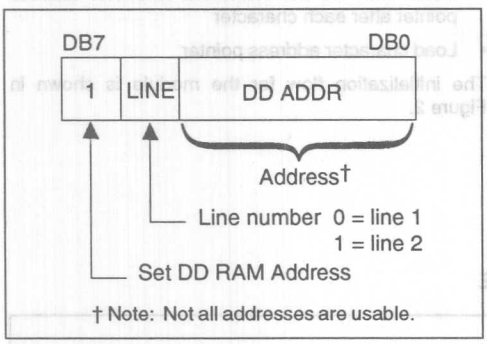
3

FIGURE 2: INITIALIZATION FLOW FOR LCD MODULE



After initialization, each character address is individually addressable. Figure 3 shows the structure of the command to specify the character address.

FIGURE 3: CHARACTER ADDRESS COMMAND FORMAT



The Hatachi Display Drive (HD44780A) has 80 bytes of RAM. The LM032L modules only use 40 bytes of the available RAM (2 x 20 characters). It is possible to use the remaining RAM locations for storage of other information.

Figure 4 shows the display data positions supported by the display driver as well as the characters actually displayed by the module (the shaded addresses).

The program example implemented here uses the auto character increment feature. This automatically increments the character address pointer after each character is written to the display.

CONCLUSION

The Hitachi LM032L character display module is useful for the display of information. The selection of 4-bit or 8-bit data transfer mode is strictly a program memory size / I/O resource trade-off. The supplied code is easily used in one of three common data interfaces. The source is easily modifiable to the designers specific application needs. Other display modules / drivers maybe implemented with the appropriate modifications. Table 4 shows the resource requirements for the three subroutines SEND_CHAR, SEND_COMMAND, and BUSY_CHECK in the various data interface modes.

TABLE 4: RESOURCE REQUIREMENTS

Mode	Program Memory	Data Memory	Verified On
8-bit	32	3	PICDEM-2 *
4-bit, Data transferred on the high nibble of the port	53	3	PICDEM-2 *
4-bit, Data transferred on the high nibble of the port	53	3	Low Power Real Time Clock Board (AN582)

* Jumper J6 must be removed.

FIGURE 4: DISPLAY DRIVER (DD) RAM LOCATIONS

digit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	·····	33	34	35	36	37	38	39	40	← Display position
1-line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	·····	20	21	22	23	24	25	26	27	← DD RAM address (Hexadecimal)
2-line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	·····	60	61	62	63	64	65	66	67	←

Note: Shaded locations are displayed on the LM032L display module.

Written By: Mark Palmer - Sr. Application Engineer
Code By: Mark Palmer / Scott Fink - Sr. Application Engineers

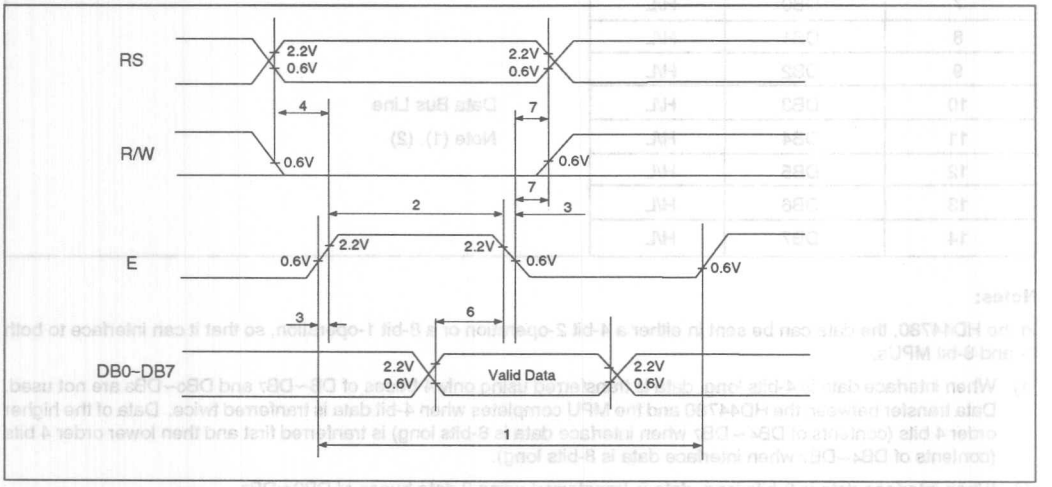
© 1994 Microchip Technology Inc.

APPENDIX A: LM032L TIMING REQUIREMENTS

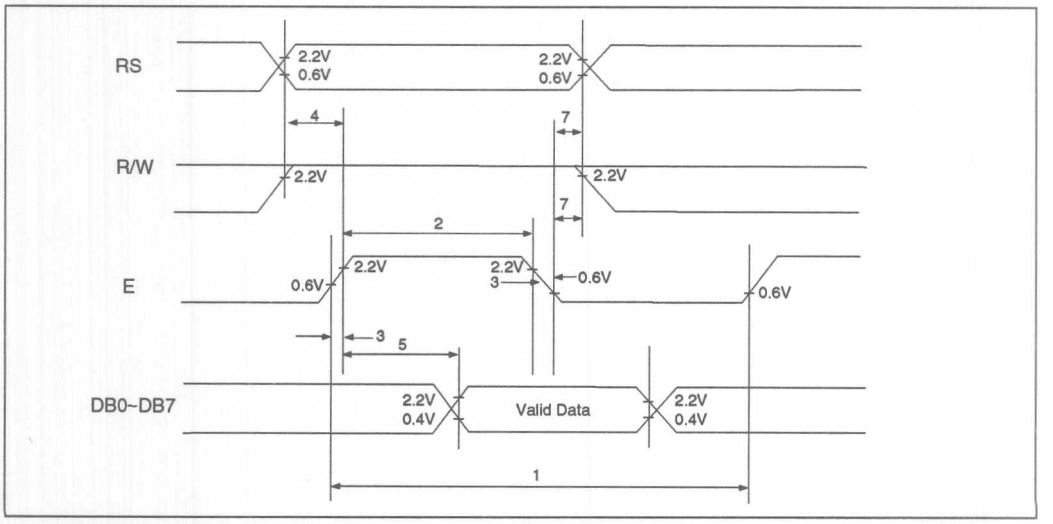
TIMING CHARACTERISTICS

Parm #	Symbol	Characterisitics	Min.	Typ.	Max.	Unit
1	t_{cyc}	Enable cycle time	1.0	—	—	µs
2	PW_{EH}	Enable pulse width	450	—	—	ns
3	t_{Er}, t_{Ef}	Enable rise / fall time	—	—	25	ns
4	t_{AS}	RS, R/W set up time	140	—	—	ns
5	t_{DDR}	Data delay time	—	—	320	ns
6	t_{DSW}	Data set up time	195	—	—	ns
7	t_H	Hold time	20	—	—	ns

DATA WRITE INTERFACE TIMING



DATA READ INTERFACE TIMING



Note: Refer to Hitachi documentation for most current timing specifications.

© 1994 Microchip Technology Inc.

LM032L PIN CONNECTION

Pin No.	Symbol	Level	Function	
1	Vss	—	0V	
2	VDD	—	+5V	Power Supply
3	Vo	—	—	
4	RS	H/L	L: Instruction Code Input H: Data Input	
5	R/W	H/L	H: Data Read (LCD module→MPU) L: Data Write (LCD module←MPU)	
6	E	H,H→L	Enable Signal	
7	DB0	H/L	Data Bus Line Note (1), (2)	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L		
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

Notes:

In the HD44780, the data can be sent in either a 4-bit 2-operation or a 8-bit 1-operation, so that it can interface to both 4- and 8-bit MPUs.

(1) When interface data is 4-bits long, data is transferred using only 4 buses of DB~DB7 and DB0~DB3 are not used. Data transfer between the HD44780 and the MPU completes when 4-bit data is tranferred twice. Data of the higher order 4 bits (contents of DB4 ~ DB7 when interface data is 8-bits long) is tranferred first and then lower order 4 bits (contents of DB4~DB7 when interface data is 8-bits long).

(2) When interface data is 8-bits long, data is transferred using 8 data buses of DB0~DB7.

© 1994 Microchip Technology Inc.

APPENDIX B: 8-BIT DATA INTERFACE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:53:47                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;
                     0012 ;
                     0013         include <C74_reg.h>
                     0233
                     0013
                     0014         include <lm0321.h>
                     0014
                     0015 ;
0000                 0016 Four_bit       EQU     FALSE       ; Selects 4- or 8-bit data transfers
0001                 0017 Data_HI        EQU     TRUE       ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;
                     0019 ;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;
                     0022 LCD_DATA       EQU     PORTB
                     0023 LCD_DATA_TRIS  EQU     TRISB
                     0024 ;
                     0025     else
                     0026 ;
0008                 0027 LCD_DATA       EQU     PORTD
0088                 0028 LCD_DATA_TRIS  EQU     TRISD
                     0029 ;
                     0030     endif
                     0031 ;
```


© 1994 Microchip Technology Inc.

```
0005                    0032 LCD_CNTL    EQU     PORTA
                        0033 ;
                        0034 ;
                        0035 ;
                        0036 ; LCD Display Commands and Control Signal names.
                        0037 ;
                        0038     if ( Four_bit && !Data_HI )
                        0039 ;
                        0040 E           EQU     0               ; LCD Enable control line
                        0041 R_W         EQU     1               ; LCD Read/Write control line
                        0042 RS          EQU     2               ; LCD Register Select control line
                        0043 ;
                        0044     else
                        0045 ;
0003                    0046 E           EQU     3               ; LCD Enable control line
0002                    0047 R_W         EQU     2               ; LCD Read/Write control line
0001                    0048 RS          EQU     1               ; LCD Register Select control line
                        0049 ;
                        0050     endif
                        0051 ;
                        0052 ;
0030                    0053 TEMP1       EQU     0x030
                        0054 ;
                        0055         org     RESET_V             ; RESET vector location
0000 2808               0056 RESET       GOTO    START           ;
                        0057 ;
                        0058 ; This is the Periperal Interrupt routine. Should NOT get here
                        0059 ;
                        0061         org     ISR_V               ; Interrupt vector location
                        0062 PER_INT_V
0004 1283               0063 ERROR1      BCF     STATUS, RP0     ; Bank 0
0005 1407               0064             BSF     PORTC, 0
0006 1007               0065             BCF     PORTC, 0
0007 2804               0066             GOTO    ERROR1
                        0067 ;
                        0068 ;
                        0069 ;
                        0070 START                               ; POWER_ON Reset (Beginning of program)
0008 0183               0071             CLRF    STATUS          ; Do initialization (Bank 0)
0009 018B               0072             CLRF    INTCON
000A 018C               0073             CLRF    PIR1
000B 1683               0074             BSF     STATUS, RP0     ; Bank 1
000C 3000               0075             MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
000D 0081               0076             MOVWF   OPTION_R        ;
```


© 1994 Microchip Technology Inc.

```
000E 018C  0077            CLRF    PIE1            ; Disable all peripheral interrupts
000F 30FF  0078            MOVLW   0xFF            ;
0010 009F  0079            MOVWF   ADCON1          ; Port A is Digital.
           0080 ;
           0081 ;
0011 1283  0082            BCF     STATUS, RP0     ; Bank 0
0012 0185  0083            CLRF    PORTA           ; ALL PORT output should output Low.
0013 0186  0084            CLRF    PORTB
0014 0187  0085            CLRF    PORTC
0015 0188  0086            CLRF    PORTD
0016 0189  0087            CLRF    PORTE
0017 1010  0088            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
           0089 ;
0018 1683  0090            BSF     STATUS, RP0     ; Select Bank 1
0019 0185  0091            CLRF    TRISA           ; RA5 -  0 outputs
001A 30F0  0092            MOVLW   0xF0            ;
001B 0086  0093            MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187  0094            CLRF    TRISC           ; RC Port are outputs
001D 1407  0095            BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188  0096            CLRF    TRISD           ; RD Port are outputs
001F 0189  0097            CLRF    TRISE           ; RE Port are outputs
0020 140C  0098            BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781  0099            BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283  0100            BCF     STATUS, RP0     ; Select Bank 0
           0101 ;
           0103 ;
           0104 ; Initilize the LCD Display Module
           0105 ;
0023 0185  0106            CLRF    LCD_CNTL        ; ALL PORT output should output Low.
           0107
           0108 DISPLAY_INIT
           0109     if ( Four_bit && !Data_HI )
           0110             MOVLW   0x02            ; Command for 4-bit interface low nibble
           0111     endif
           0112 ;
           0113     if ( Four_bit && Data_HI )
           0114             MOVLW   0x020           ; Command for 4-bit interface high nibble
           0115     endif
           0116 ;
           0117     if ( !Four_bit )
0024 3038  0118             MOVLW   0x038           ; Command for 8-bit interface
           0119     endif
           0120 ;
0025 0088  0121             MOVWF   LCD_DATA        ;
0026 1585  0122             BSF     LCD_CNTL, E     ;
```

© 1994 Microchip Technology Inc.

```
0027 1185          0123            BCF     LCD_CNTL, E     ;
                   0124 ;
                   0125 ; This routine takes the calculated times that the delay loop needs to
                   0126 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                   0127 ; frequency of operation. These uses registers before they are needed to
                   0128 ; store the time.
                   0129 ;
0028 3006          0130 LCD_DELAY  MOVLW   LCD_INIT_DELAY  ;
0029 00B3          0131            MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
002A 01B4          0132            CLRF    LSD             ;
002B 0BB4          0133 LOOP2      DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002C 282B          0134            GOTO    LOOP2           ;
002D 0BB3          0135            DECFSZ  MSD             ;
                   0136 END_LCD_DELAY
002E 282B          0137            GOTO    LOOP2           ;
                   0138 ;
                   0139 ; Command sequence for 2 lines of 5x7 characters
                   0140 ;
                   0141 CMD_SEQ
                   0142 ;
                   0143    if ( Four_bit )
                   0144        if ( !Data_HI )
                   0145            MOVLW   0X02            ; 4-bit low nibble xfer
                   0146        else
                   0147            MOVLW   0X020           ; 4-bit high nibble xfer
                   0148        endif
                   0149 ;
                   0150    else                            ; 8-bit mode
002F 3038          0151            MOVLW   0X038
                   0152    endif
                   0153 ;
0030 0088          0154            MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
0031 1585          0155            BSF     LCD_CNTL, E     ;
0032 1185          0156            BCF     LCD_CNTL, E     ;
                   0157 ;
                   0158    if ( Four_bit )                 ; This code for only 4-bit mode (2nd xfer)
                   0159        if ( !Data_HI )
                   0160            MOVLW   0x08            ; 4-bit low nibble xfer
                   0161        else
                   0162            MOVLW   0x080           ; 4-bit high nibble xfer
                   0163        endif
                   0164            MOVWF   LCD_DATA        ;
                   0165            BSF     LCD_CNTL, E     ;
                   0166            BCF     LCD_CNTL, E     ;
```


© 1994 Microchip Technology Inc.

```
                        0167     endif
                        0168 ;
                        0169 ; Busy Flag should be valid after this point
                        0170 ;
0033 300C               0171             MOVLW   DISP_ON          ;
0034 206A               0172             CALL    SEND_CMD         ;
0035 3001               0173             MOVLW   CLR_DISP         ;
0036 206A               0174             CALL    SEND_CMD         ;
0037 3006               0175             MOVLW   ENTRY_INC        ;
0038 206A               0176             CALL    SEND_CMD         ;
0039 3080               0177             MOVLW   DD_RAM_ADDR      ;
003A 206A               0178             CALL    SEND_CMD         ;
                        0179 ;
                        0181 ;
                        0182 ;Send a message the hard way
003B 304D               0183             movlw   'M'
003C 2061               0184             call    SEND_CHAR
003D 3069               0185             movlw   'i'
003E 2061               0186             call    SEND_CHAR
003F 3063               0187             movlw   'c'
0040 2061               0188             call    SEND_CHAR
0041 3072               0189             movlw   'r'
0042 2061               0190             call    SEND_CHAR
0043 306F               0191             movlw   'o'
0044 2061               0192             call    SEND_CHAR
0045 3063               0193             movlw   'c'
0046 2061               0194             call    SEND_CHAR
0047 3068               0195             movlw   'h'
0048 2061               0196             call    SEND_CHAR
0049 3069               0197             movlw   'i'
004A 2061               0198             call    SEND_CHAR
004B 3070               0199             movlw   'p'
004C 2061               0200             call    SEND_CHAR
                        0201
004D 30C0               0202             movlw   B'11000000'      ;Address DDRam first character, second line
004E 206A               0203             call    SEND_CMD
                        0204
                        0205 ;Demonstration of the use of a table to output a message
004F 3000               0206             movlw   0                ;Table address of start of message
                        0207 dispmsg
0050 00B0               0208             movwf   TEMP1            ;TEMP1 holds start of message address
0051 2085               0209             call    Table
0052 39FF               0210             andlw   0FFh             ;Check if at end of message (zero
0053 1903               0211             btfsc   STATUS,Z         ;returned at end)
0054 2859               0212             goto    out
0055 2061               0213             call    SEND_CHAR        ;Display character
0056 0830               0214             movf    TEMP1,w          ;Point to next character
```


© 1994 Microchip Technology Inc.

```
0057 3E01   0215            addlw   1
0058 2850   0216            goto    dispmsg
            0217 out
            0218 loop
0059 2859   0219            goto    loop            ;Stay here forever
            0220 ;
            0221 ;
            0222 INIT_DISPLAY
005A 300C   0223            MOVLW   DISP_ON           ; Display On, Cursor On
005B 206A   0224            CALL    SEND_CMD          ; Send This command to the Display Module
005C 3001   0225            MOVLW   CLR_DISP          ; Clear the Display
005D 206A   0226            CALL    SEND_CMD          ; Send This command to the Display Module
005E 3006   0227            MOVLW   ENTRY_INC         ; Set Entry Mode Inc., No shift
005F 206A   0228            CALL    SEND_CMD          ; Send This command to the Display Module
0060 0008   0229            RETURN
            0230 ;
            0232 ;
            0233 ;*****************************************************************
            0234 ;* The LCD Module Subroutines                                    *
            0235 ;*****************************************************************
            0236 ;
            0237     if ( Four_bit )      ; 4-bit Data transfers?
            0238 ;
            0239         if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
            0240 ;
            0241 ;*****************************************************************
            0242 ;*SendChar - Sends character to LCD                             *
            0243 ;*This routine splits the character into the upper and lower    *
            0244 ;*nibbles and sends them to the LCD, upper nibble first.        *
            0245 ;*****************************************************************
            0246 ;
            0247 SEND_CHAR
            0248            MOVWF   CHAR            ;Character to be sent is in W
            0249            CALL    BUSY_CHECK      ;Wait for LCD to be ready
            0250            MOVF    CHAR, w
            0251            ANDLW   0xF0            ;Get upper nibble
            0252            MOVWF   LCD_DATA        ;Send data to LCD
            0253            BCF     LCD_CNTL, R_W   ;Set LCD to read
            0254            BSF     LCD_CNTL, RS    ;Set LCD to data mode
            0255            BSF     LCD_CNTL, E     ;toggle E for LCD
            0256            BCF     LCD_CNTL, E
            0257            SWAPF   CHAR, w
            0258            ANDLW   0xF0            ;Get lower nibble
            0259            MOVWF   LCD_DATA        ;Send data to LCD
            0260            BSF     LCD_CNTL, E     ;toggle E for LCD
            0261            BCF     LCD_CNTL, E
```


© 1994 Microchip Technology Inc.

```
                0262             RETURN
                0263 ;
                0264         else            ; 4-bit transfers on the low nibble of the PORT
                0265 ;
                0266 ;*****************************************************************
                0267 ;* SEND_CHAR - Sends character to LCD                            *
                0268 ;* This routine splits the character into the upper and lower    *
                0269 ;* nibbles and sends them to the LCD, upper nibble first.        *
                0270 ;* The data is transmitted on the PORT<3:0> pins                 *
                0271 ;*****************************************************************
                0272 ;
                0273 SEND_CHAR
                0274             MOVWF   CHAR            ; Character to be sent is in W
                0275             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                0276             SWAPF   CHAR, W
                0277             ANDLW   0x0F            ; Get upper nibble
                0278             MOVWF   LCD_DATA        ; Send data to LCD
                0279             BCF     LCD_CNTL, R_W   ; Set LCD to read
                0280             BSF     LCD_CNTL, RS    ; Set LCD to data mode
                0281             BSF     LCD_CNTL, E     ; toggle E for LCD
                0282             BCF     LCD_CNTL, E
                0283             MOVF    CHAR, W
                0284             ANDLW   0x0F            ; Get lower nibble
                0285             MOVWF   LCD_DATA        ; Send data to LCD
                0286             BSF     LCD_CNTL, E     ; toggle E for LCD
                0287             BCF     LCD_CNTL, E
                0288             RETURN
                0289 ;
                0290         endif
                0291     else
                0292 ;
                0293 ;*****************************************************************
                0294 ;* SEND_CHAR - Sends character contained in register W to LCD    *
                0295 ;* This routine sends the entire character to the PORT           *
                0296 ;* The data is transmitted on the PORT<7:0> pins                 *
                0297 ;*****************************************************************
                0298 ;
                0299 SEND_CHAR
0061 00B6       0300             MOVWF   CHAR            ; Character to be sent is in W
0062 2073       0301             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0063 0836       0302             MOVF    CHAR, w
0064 0088       0303             MOVWF   LCD_DATA        ; Send data to LCD
0065 1105       0304             BCF     LCD_CNTL, R_W   ; Set LCD in read mode
0066 1485       0305             BSF     LCD_CNTL, RS    ; Set LCD in data mode
0067 1585       0306             BSF     LCD_CNTL, E     ; toggle E for LCD
0068 1185       0307             BCF     LCD_CNTL, E
0069 0008       0308             RETURN
```

© 1994 Microchip Technology Inc.

```
0309 ;
0310     endif
0311 ;
0313 ;
0314 ;*********************************************************************
0315 ;* SendCmd - Sends command to LCD                                    *
0316 ;* This routine splits the command into the upper and lower          *
0317 ;* nibbles and sends them to the LCD, upper nibble first.            *
0318 ;* The data is transmitted on the PORT<3:0> pins                     *
0319 ;*********************************************************************
0320 ;
0321     if ( Four_bit )       ; 4-bit Data transfers?
0322 ;
0323         if ( Data_HI )    ; 4-bit transfers on the high nibble of the PORT
0324 ;
0325 ;*********************************************************************
0326 ;* SEND_CMD - Sends command to LCD                                   *
0327 ;* This routine splits the command into the upper and lower          *
0328 ;* nibbles and sends them to the LCD, upper nibble first.            *
0329 ;*********************************************************************
0330
0331 SEND_CMD
0332             MOVWF   CHAR            ; Character to be sent is in W
0333             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0334             MOVF    CHAR,w
0335             ANDLW   0xF0            ; Get upper nibble
0336             MOVWF   LCD_DATA        ; Send data to LCD
0337             BCF     LCD_CNTL,R_W    ; Set LCD to read
0338             BCF     LCD_CNTL,RS     ; Set LCD to command mode
0339             BSF     LCD_CNTL,E      ; toggle E for LCD
0340             BCF     LCD_CNTL,E
0341             SWAPF   CHAR,w
0342             ANDLW   0xF0            ; Get lower nibble
0343             MOVWF   LCD_DATA        ; Send data to LCD
0344             BSF     LCD_CNTL,E      ; toggle E for LCD
0345             BCF     LCD_CNTL,E
0346             RETURN
0347 ;
0348     else            ; 4-bit transfers on the low nibble of the PORT
0349 ;
0350 SEND_CMD
0351             MOVWF   CHAR            ; Character to be sent is in W
0352             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0353             SWAPF   CHAR, W
0354             ANDLW   0x0F            ; Get upper nibble
0355             MOVWF   LCD_DATA        ; Send data to LCD
0356             BCF     LCD_CNTL, R_W   ; Set LCD to read
```


© 1994 Microchip Technology Inc.

```
                0357              BCF     LCD_CNTL, RS    ; Set LCD to command mode
                0358              BSF     LCD_CNTL, E     ; toggle E for LCD
                0359              BCF     LCD_CNTL, E
                0360              MOVF    CHAR, W
                0361              ANDLW   0x0F            ; Get lower nibble
                0362              MOVWF   LCD_DATA        ; Send data to LCD
                0363              BSF     LCD_CNTL, E     ; toggle E for LCD
                0364              BCF     LCD_CNTL, E
                0365              RETURN
                0366 ;
                0367          endif
                0368      else
                0369 ;
                0370 ;*****************************************************************
                0371 ;* SEND_CND - Sends command contained in register W to LCD     *
                0372 ;* This routine sends the entire character to the PORT         *
                0373 ;* The data is transmitted on the PORT<7:0> pins               *
                0374 ;*****************************************************************
                0375
                0376 SEND_CMD
006A 00B6       0377              MOVWF   CHAR            ; Command to be sent is in W
006B 2073       0378              CALL    BUSY_CHECK      ; Wait for LCD to be ready
006C 0836       0379              MOVF    CHAR, w
006D 0088       0380              MOVWF   LCD_DATA        ; Send data to LCD
006E 1105       0381              BCF     LCD_CNTL, R_W   ; Set LCD in read mode
006F 1085       0382              BCF     LCD_CNTL, RS    ; Set LCD in command mode
0070 1585       0383              BSF     LCD_CNTL, E     ; toggle E for LCD
0071 1185       0384              BCF     LCD_CNTL, E
0072 0008       0385              RETURN
                0386 ;
                0387      endif
                0388 ;
                0390 ;
                0391      if ( Four_bit )       ; 4-bit Data transfers?
                0392 ;
                0393          if ( Data_HI )    ; 4-bit transfers on the high nibble of the PORT
                0394 ;
                0395 ;*****************************************************************
                0396 ;* This routine checks the busy flag, returns when not busy      *
                0397 ;*  Affects:                                                     *
                0398 ;*      TEMP - Returned with busy/address                        *
                0399 ;*****************************************************************
                0400 ;
                0401 BUSY_CHECK
                0402              BSF     STATUS, RP0     ; Select Register page 1
                0403              MOVLW   0xFF            ; Set Port_D for input
                0404              MOVWF   LCD_DATA_TRIS
```

© 1994 Microchip Technology Inc.

```
0405            BCF     STATUS, RP0     ; Select Register page 0
0406            BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0407            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0408            BSF     LCD_CNTL, E     ; Set E high
0409            BCF     LCD_CNTL, E     ; Set E low
0410            MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0411            ANDLW   0xF0            ; Mask out lower nibble
0412            MOVWF   TEMP
0413            BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0414            BCF     LCD_CNTL, E
0415            SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
0416            ANDLW   0x0F            ; Mask out upper nibble
0417            IORWF   TEMP            ; Combine nibbles
0418            BTFSC   TEMP, 7         ; Check busy flag, high = busy
0419            GOTO    BUSY_CHECK      ; If busy, check again
0420            BCF     LCD_CNTL, R_W
0421            BSF     STATUS, RP0     ; Select Register page 1
0422            MOVLW   0x0F
0423            MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
0424            BCF     STATUS, RP0     ; Select Register page 0
0425            RETURN
0426 ;
0427        else            ; 4-bit transfers on the low nibble of the PORT
0428 ;
0429 ;*****************************************************************
0430 ;* This routine checks the busy flag, returns when not busy      *
0431 ;*  Affects:                                                     *
0432 ;*      TEMP - Returned with busy/address                        *
0433 ;*****************************************************************
0434 ;
0435 BUSY_CHECK
0436            BSF     STATUS, RP0     ; Bank 1
0437            MOVLW   0xFF            ; Set PortB for input
0438            MOVWF   LCD_DATA_TRIS
0439            BCF     STATUS, RP0     ; Bank 0
0440            BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0441            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0442            BSF     LCD_CNTL, E     ; Set E high
0443            BCF     LCD_CNTL, E     ; Set E low
0444            SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0445            ANDLW   0xF0            ; Mask out lower nibble
0446            MOVWF   TEMP            ;
0447            BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0448            BCF     LCD_CNTL, E
0449            MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
0450            ANDLW   0x0F            ; Mask out upper nibble
0451            IORWF   TEMP, F         ; Combine nibbles
```


© 1994 Microchip Technology Inc.

```
                    0452             BTFSC   TEMP, 7          ; Check busy flag, high = busy
                    0453             GOTO    BUSY_CHECK       ; If busy, check again
                    0454             BCF     LCD_CNTL, R_W
                    0455             BSF     STATUS, RP0      ; Bank 1
                    0456             MOVLW   0xF0             ;
                    0457             MOVWF   LCD_DATA_TRIS    ; RB7 - 4 = inputs, RB3 - 0 = output
                    0458             BCF     STATUS, RP0      ; Bank 0
                    0459             RETURN
                    0460 ;
                    0461         endif
                    0462     else
                    0463 ;
                    0464 ;*****************************************************************
                    0465 ;* This routine checks the busy flag, returns when not busy    *
                    0466 ;*  Affects:                                                     *
                    0467 ;*       TEMP - Returned with busy/address                      *
                    0468 ;*****************************************************************
                    0469 ;
                    0470 BUSY_CHECK
0073 1683           0471             BSF     STATUS,RP0       ; Select Register page 1
0074 30FF           0472             MOVLW   0xFF             ; Set port_D for input
0075 0088           0473             MOVWF   LCD_DATA_TRIS
0076 1283           0474             BCF     STATUS, RP0      ; Select Register page 0
0077 1085           0475             BCF     LCD_CNTL, RS     ; Set LCD for command mode
0078 1505           0476             BSF     LCD_CNTL, R_W    ; Setup to read busy flag
0079 1585           0477             BSF     LCD_CNTL, E      ; Set E high
007A 1185           0478             BCF     LCD_CNTL, E      ; Set E low
007B 0808           0479             MOVF    LCD_DATA, w      ; Read busy flag, DDram address
007C 00B5           0480             MOVWF   TEMP
007D 1BB5           0481             BTFSC   TEMP, 7          ; Check busy flag, high=busy
007E 2873           0482             GOTO    BUSY_CHECK
007F 1105           0483             BCF     LCD_CNTL, R_W
0080 1683           0484             BSF     STATUS, RP0      ; Select Register page 1
0081 3000           0485             MOVLW   0x00
0082 0088           0486             MOVWF   LCD_DATA_TRIS    ; Set port_D for output
0083 1283           0487             BCF     STATUS, RP0      ; Select Register page 0
0084 0008           0488             RETURN
                    0489 ;
                    0490     endif
                    0492 ;
                    0493 Table
0085 0782           0494             addwf   PCL       ;Jump to char pointed to in W reg
0086 344D           0495             retlw   'M'
0087 3469           0496             retlw   'i'
0088 3463           0497             retlw   'c'
0089 3472           0498             retlw   'r'
008A 346F           0499             retlw   'o'
008B 3463           0500             retlw   'c'
```

© 1994 Microchip Technology Inc.

```
008C 3468               0501            retlw   'h'
008D 3469               0502            retlw   'i'
008E 3470               0503            retlw   'p'
008F 3420               0504            retlw   ' '
0090 3454               0505            retlw   'T'
0091 3465               0506            retlw   'e'
0092 3463               0507            retlw   'c'
0093 3468               0508            retlw   'h'
0094 346E               0509            retlw   'n'
0095 346F               0510            retlw   'o'
0096 346C               0511            retlw   'l'
0097 346F               0512            retlw   'o'
0098 3467               0513            retlw   'g'
0099 3479               0514            retlw   'y'
                        0515 Table_End
009A 3400               0516            retlw   0
                        0517 ;
                        0518        if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                        0519          MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
                        0520        endif
                        0521 ;
                        0522
                        0523
                        0524
                        0525        end
                        0526
                        0527
                        0528
                        0529
                        0530
```


© 1994 Microchip Technology Inc.

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:53:47                PAGE 14

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXX-- -------- --------
00C0 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :    13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

© 1994 Microchip Technology Inc.

APPENDIX C: 4-BIT DATA INTERFACE, HIGH NIBBLE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:59:12                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;
                     0012 ;
                     0013         include <C74_reg.h>
                     0233
                     0013
                     0014         include <lm032l.h>
                     0014
                     0015 ;
0001                 0016 Four_bit         EQU     TRUE       ; Selects 4- or 8-bit data transfers
0001                 0017 Data_HI          EQU     TRUE       ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;
                     0019 ;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;
                     0022 LCD_DATA         EQU     PORTB
                     0023 LCD_DATA_TRIS    EQU     TRISB
                     0024 ;
                     0025     else
                     0026 ;
0008                 0027 LCD_DATA         EQU     PORTD
0088                 0028 LCD_DATA_TRIS    EQU     TRISD
                     0029 ;
                     0030     endif
                     0031 ;
0005                 0032 LCD_CNTL         EQU     PORTA
                     0033 ;
                     0034 ;
```


© 1994 Microchip Technology Inc.

```
                    0035 ;
                    0036 ; LCD Display Commands and Control Signal names.
                    0037 ;
                    0038     if ( Four_bit && !Data_HI )
                    0039 ;
                    0040 E                          EQU             0               ; LCD Enable control line
                    0041 R_W                        EQU             1               ; LCD Read/Write control line
                    0042 RS                         EQU             2               ; LCD Register Select control
line
                    0043 ;
                    0044     else
                    0045 ;
0003                0046 E                          EQU             3               ; LCD Enable control line
0002                0047 R_W                        EQU             2               ; LCD Read/Write control line
0001                0048 RS                         EQU             1               ; LCD Register Select control
line
                    0049 ;
                    0050     endif
                    0051 ;
                    0052 ;
0030                0053 TEMP1      EQU     0x030
                    0054 ;
                    0055         org     RESET_V             ; RESET vector location
0000 2808           0056 RESET      GOTO    START               ;
                    0057 ;
                    0058 ; This is the Peripheral Interrupt routine. Should NOT get here
                    0059 ;
                    0061         org     ISR_V           ; Interrupt vector location
                    0062 PER_INT_V
0004 1283           0063 ERROR1     BCF     STATUS, RP0         ; Bank 0
0005 1407           0064            BSF     PORTC, 0
0006 1007           0065            BCF     PORTC, 0
0007 2804           0066            GOTO    ERROR1
                    0067 ;
                    0068 ;
                    0069 ;
                    0070 START                          ; POWER_ON Reset (Beginning of program)
0008 0183           0071            CLRF    STATUS          ; Do initialization (Bank 0)
0009 018B           0072            CLRF    INTCON
000A 018C           0073            CLRF    PIR1
000B 1683           0074            BSF     STATUS, RP0     ; Bank 1
000C 3000           0075            MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
000D 0081           0076            MOVWF   OPTION_R        ;
000E 018C           0077            CLRF    PIE1            ; Disable all peripheral interrupts
000F 30FF           0078            MOVLW   0xFF            ;
0010 009F           0079            MOVWF   ADCON1          ; Port A is Digital.
                    0080 ;
                    0081 ;
```

© 1994 Microchip Technology Inc.

```
0011 1283           0082            BCF     STATUS, RP0     ; Bank 0
0012 0185           0083            CLRF    PORTA           ; ALL PORT output should output Low.
0013 0186           0084            CLRF    PORTB
0014 0187           0085            CLRF    PORTC
0015 0188           0086            CLRF    PORTD
0016 0189           0087            CLRF    PORTE
0017 1010           0088            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                    0089 ;
0018 1683           0090            BSF     STATUS, RP0     ; Select Bank 1
0019 0185           0091            CLRF    TRISA           ; RA5 -  0 outputs
001A 30F0           0092            MOVLW   0xF0            ;
001B 0086           0093            MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187           0094            CLRF    TRISC           ; RC Port are outputs
001D 1407           0095            BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188           0096            CLRF    TRISD           ; RD Port are outputs
001F 0189           0097            CLRF    TRISE           ; RE Port are outputs
0020 140C           0098            BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781           0099            BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283           0100            BCF     STATUS, RP0     ; Select Bank 0
                    0101 ;
                    0103 ;
                    0104 ; Initilize the LCD Display Module
                    0105 ;
0023 0185           0106            CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                    0107
                    0108 DISPLAY_INIT
                    0109    if ( Four_bit && !Data_HI )
                    0110            MOVLW   0x02            ; Command for 4-bit interface low nibble
                    0111    endif
                    0112 ;
                    0113    if ( Four_bit && Data_HI )
0024 3020           0114            MOVLW   0x020           ; Command for 4-bit interface high nibble
                    0115    endif
                    0116 ;
                    0117    if ( !Four_bit )
                    0118            MOVLW   0x038           ; Command for 8-bit interface
                    0119    endif
                    0120 ;
0025 0088           0121            MOVWF   LCD_DATA        ;
0026 1585           0122            BSF     LCD_CNTL, E     ;
0027 1185           0123            BCF     LCD_CNTL, E     ;
                    0124 ;
                    0125 ; This routine takes the calculated times that the delay loop needs to
                    0126 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                    0127 ; frequency of operation. These uses registers before they are needed to
                    0128 ; store the time.
                    0129 ;
0028 3006           0130 LCD_DELAY  MOVLW   LCD_INIT_DELAY  ;
```


© 1994 Microchip Technology Inc.

```
0029 00B3          0131             MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
002A 01B4          0132             CLRF    LSD             ;
002B 0BB4          0133 LOOP2       DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002C 282B          0134             GOTO    LOOP2           ;
002D 0BB3          0135             DECFSZ  MSD             ;
                   0136 END_LCD_DELAY
002E 282B          0137             GOTO    LOOP2           ;
                   0138 ;
                   0139 ; Command sequence for 2 lines of 5x7 characters
                   0140 ;
                   0141 CMD_SEQ
                   0142 ;
                   0143     if ( Four_bit )
                   0144         if ( !Data_HI )
                   0145             MOVLW   0X02            ; 4-bit low nibble xfer
                   0146         else
002F 3020          0147             MOVLW   0X020           ; 4-bit high nibble xfer
                   0148         endif
                   0149 ;
                   0150     else                            ; 8-bit mode
                   0151             MOVLW   0X038
                   0152     endif
                   0153 ;
0030 0088          0154             MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
0031 1585          0155             BSF     LCD_CNTL, E     ;
0032 1185          0156             BCF     LCD_CNTL, E     ;
                   0157 ;
                   0158     if ( Four_bit )                 ; This code for only 4-bit mode (2nd xfer)
                   0159         if ( !Data_HI )
                   0160             MOVLW   0x08            ; 4-bit low nibble xfer
                   0161         else
0033 3080          0162             MOVLW   0x080           ; 4-bit high nibble xfer
                   0163         endif
0034 0088          0164             MOVWF   LCD_DATA        ;
0035 1585          0165             BSF     LCD_CNTL, E     ;
0036 1185          0166             BCF     LCD_CNTL, E     ;
                   0167     endif
                   0168 ;
                   0169 ; Busy Flag should be valid after this point
                   0170 ;
0037 300C          0171             MOVLW   DISP_ON         ;
0038 2074          0172             CALL    SEND_CMD        ;
0039 3001          0173             MOVLW   CLR_DISP        ;
003A 2074          0174             CALL    SEND_CMD        ;
003B 3006          0175             MOVLW   ENTRY_INC       ;
003C 2074          0176             CALL    SEND_CMD        ;
003D 3080          0177             MOVLW   DD_RAM_ADDR     ;
003E 2074          0178             CALL    SEND_CMD        ;
```

© 1994 Microchip Technology Inc.

```
                        0179 ;
                        0181 ;
                        0182 ;Send a message the hard way
003F 304D               0183            movlw   'M'
0040 2065               0184            call    SEND_CHAR
0041 3069               0185            movlw   'i'
0042 2065               0186            call    SEND_CHAR
0043 3063               0187            movlw   'c'
0044 2065               0188            call    SEND_CHAR
0045 3072               0189            movlw   'r'
0046 2065               0190            call    SEND_CHAR
0047 306F               0191            movlw   'o'
0048 2065               0192            call    SEND_CHAR
0049 3063               0193            movlw   'c'
004A 2065               0194            call    SEND_CHAR
004B 3068               0195            movlw   'h'
004C 2065               0196            call    SEND_CHAR
004D 3069               0197            movlw   'i'
004E 2065               0198            call    SEND_CHAR
004F 3070               0199            movlw   'p'
0050 2065               0200            call    SEND_CHAR
                        0201
0051 30C0               0202            movlw   B'11000000'     ;Address DDRam first character, second line
0052 2074               0203            call    SEND_CMD
                        0204
                        0205 ;Demonstration of the use of a table to output a message
0053 3000               0206            movlw   0               ;Table address of start of message
                        0207 dispmsg
0054 00B0               0208            movwf   TEMP1           ;TEMP1 holds start of message address
0055 209B               0209            call    Table
0056 39FF               0210            andlw   0FFh            ;Check if at end of message (zero
0057 1903               0211            btfsc   STATUS,Z        ;returned at end)
0058 285D               0212            goto    out
0059 2065               0213            call    SEND_CHAR       ;Display character
005A 0830               0214            movf    TEMP1,w         ;Point to next character
005B 3E01               0215            addlw   1
005C 2854               0216            goto    dispmsg
                        0217 out
                        0218 loop
005D 285D               0219            goto    loop            ;Stay here forever
                        0220 ;
                        0221 ;
                        0222 INIT_DISPLAY
005E 300C               0223            MOVLW   DISP_ON             ; Display On, Cursor On
005F 2074               0224            CALL    SEND_CMD            ; Send This command to the Display Module
0060 3001               0225            MOVLW   CLR_DISP            ; Clear the Display
0061 2074               0226            CALL    SEND_CMD            ; Send This command to the Display Module
0062 3006               0227            MOVLW   ENTRY_INC           ; Set Entry Mode Inc., No shift
```


© 1994 Microchip Technology Inc.

```
0063 2074           0228            CALL    SEND_CMD            ; Send This command to the Display Module
0064 0008           0229            RETURN
                    0230 ;
                    0232 ;
                    0233 ;*****************************************************************
                    0234 ;* The LCD Module Subroutines                                    *
                    0235 ;*****************************************************************
                    0236 ;
                    0237    if ( Four_bit )     ; 4-bit Data transfers?
                    0238 ;
                    0239        if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
                    0240 ;
                    0241 ;*****************************************************************
                    0242 ;*SendChar - Sends character to LCD                            *
                    0243 ;*This routine splits the character into the upper and lower   *
                    0244 ;*nibbles and sends them to the LCD, upper nibble first.       *
                    0245 ;*****************************************************************
                    0246 ;
                    0247 SEND_CHAR
0065 00B6           0248            MOVWF   CHAR            ;Character to be sent is in W
0066 2083           0249            CALL    BUSY_CHECK      ;Wait for LCD to be ready
0067 0836           0250            MOVF    CHAR, w
0068 39F0           0251            ANDLW   0xF0            ;Get upper nibble
0069 0088           0252            MOVWF   LCD_DATA        ;Send data to LCD
006A 1105           0253            BCF     LCD_CNTL, R_W   ;Set LCD to read
006B 1485           0254            BSF     LCD_CNTL, RS    ;Set LCD to data mode
006C 1585           0255            BSF     LCD_CNTL, E     ;toggle E for LCD
006D 1185           0256            BCF     LCD_CNTL, E
006E 0E36           0257            SWAPF   CHAR, w
006F 39F0           0258            ANDLW   0xF0            ;Get lower nibble
0070 0088           0259            MOVWF   LCD_DATA        ;Send data to LCD
0071 1585           0260            BSF     LCD_CNTL, E     ;toggle E for LCD
0072 1185           0261            BCF     LCD_CNTL, E
0073 0008           0262            RETURN
                    0263 ;
                    0264        else            ; 4-bit transfers on the low nibble of the PORT
                    0265 ;
                    0266 ;*****************************************************************
                    0267 ;* SEND_CHAR - Sends character to LCD                          *
                    0268 ;* This routine splits the character into the upper and lower   *
                    0269 ;* nibbles and sends them to the LCD, upper nibble first.       *
                    0270 ;* The data is transmitted on the PORT<3:0> pins                *
                    0271 ;*****************************************************************
                    0272 ;
                    0273 SEND_CHAR
                    0274            MOVWF   CHAR            ; Character to be sent is in W
                    0275            CALL    BUSY_CHECK      ; Wait for LCD to be ready
                    0276            SWAPF   CHAR, W
```

© 1994 Microchip Technology Inc.

```
0277            ANDLW   0x0F            ; Get upper nibble
0278            MOVWF   LCD_DATA        ; Send data to LCD
0279            BCF     LCD_CNTL, R_W   ; Set LCD to read
0280            BSF     LCD_CNTL, RS    ; Set LCD to data mode
0281            BSF     LCD_CNTL, E     ; toggle E for LCD
0282            BCF     LCD_CNTL, E
0283            MOVF    CHAR, W
0284            ANDLW   0x0F            ; Get lower nibble
0285            MOVWF   LCD_DATA        ; Send data to LCD
0286            BSF     LCD_CNTL, E     ; toggle E for LCD
0287            BCF     LCD_CNTL, E
0288            RETURN
0289 ;
0290        endif
0291    else
0292 ;
0293 ;*****************************************************************
0294 ;* SEND_CHAR - Sends character contained in register W to LCD    *
0295 ;* This routine sends the entire character to the PORT           *
0296 ;* The data is transmitted on the PORT<7:0> pins                 *
0297 ;*****************************************************************
0298 ;
0299 SEND_CHAR
0300            MOVWF   CHAR            ; Character to be sent is in W
0301            CALL    BUSY_CHECK      ; Wait for LCD to be ready
0302            MOVF    CHAR, w
0303            MOVWF   LCD_DATA        ; Send data to LCD
0304            BCF     LCD_CNTL, R_W   ; Set LCD in read mode
0305            BSF     LCD_CNTL, RS    ; Set LCD in data mode
0306            BSF     LCD_CNTL, E     ; toggle E for LCD
0307            BCF     LCD_CNTL, E
0308            RETURN
0309 ;
0310    endif
0311 ;
0313 ;
0314 ;*****************************************************************
0315 ;* SendCmd - Sends command to LCD                                *
0316 ;* This routine splits the command into the upper and lower      *
0317 ;* nibbles and sends them to the LCD, upper nibble first.        *
0318 ;* The data is transmitted on the PORT<3:0> pins                 *
0319 ;*****************************************************************
0320 ;
0321    if ( Four_bit )      ; 4-bit Data transfers?
0322 ;
0323        if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
0324 ;
0325 ;*****************************************************************
```


© 1994 Microchip Technology Inc.

```
                    0326 ;* SEND_CMD - Sends command to LCD                                *
                    0327 ;* This routine splits the command into the upper and lower       *
                    0328 ;* nibbles and sends them to the LCD, upper nibble first.         *
                    0329 ;*************************************************************************
                    0330
                    0331 SEND_CMD
0074 00B6           0332             MOVWF   CHAR            ; Character to be sent is in W
0075 2083           0333             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0076 0836           0334             MOVF    CHAR,w
0077 39F0           0335             ANDLW   0xF0            ; Get upper nibble
0078 0088           0336             MOVWF   LCD_DATA        ; Send data to LCD
0079 1105           0337             BCF     LCD_CNTL,R_W    ; Set LCD to read
007A 1085           0338             BCF     LCD_CNTL,RS     ; Set LCD to command mode
007B 1585           0339             BSF     LCD_CNTL,E      ; toggle E for LCD
007C 1185           0340             BCF     LCD_CNTL,E
007D 0E36           0341             SWAPF   CHAR,w
007E 39F0           0342             ANDLW   0xF0            ; Get lower nibble
007F 0088           0343             MOVWF   LCD_DATA        ; Send data to LCD
0080 1585           0344             BSF     LCD_CNTL,E      ; toggle E for LCD
0081 1185           0345             BCF     LCD_CNTL,E
0082 0008           0346             RETURN
                    0347 ;
                    0348         else            ; 4-bit transfers on the low nibble of the PORT
                    0349 ;
                    0350 SEND_CMD
                    0351             MOVWF   CHAR            ; Character to be sent is in W
                    0352             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                    0353             SWAPF   CHAR, W
                    0354             ANDLW   0x0F            ; Get upper nibble
                    0355             MOVWF   LCD_DATA        ; Send data to LCD
                    0356             BCF     LCD_CNTL, R_W   ; Set LCD to read
                    0357             BCF     LCD_CNTL, RS    ; Set LCD to command mode
                    0358             BSF     LCD_CNTL, E     ; toggle E for LCD
                    0359             BCF     LCD_CNTL, E
                    0360             MOVF    CHAR, W
                    0361             ANDLW   0x0F            ; Get lower nibble
                    0362             MOVWF   LCD_DATA        ; Send data to LCD
                    0363             BSF     LCD_CNTL, E     ; toggle E for LCD
                    0364             BCF     LCD_CNTL, E
                    0365             RETURN
                    0366 ;
                    0367         endif
                    0368     else
                    0369 ;
                    0370 ;*************************************************************
                    0371 ;* SEND_CND - Sends command contained in register W to LCD    *
                    0372 ;* This routine sends the entire character to the PORT        *
                    0373 ;* The data is transmitted on the PORT<7:0> pins              *
```

© 1994 Microchip Technology Inc.

```
                        0374 ;*****************************************************************
                        0375
                        0376 SEND_CMD
                        0377             MOVWF   CHAR            ; Command to be sent is in W
                        0378             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                        0379             MOVF    CHAR, w
                        0380             MOVWF   LCD_DATA        ; Send data to LCD
                        0381             BCF     LCD_CNTL, R_W   ; Set LCD in read mode
                        0382             BCF     LCD_CNTL, RS    ; Set LCD in command mode
                        0383             BSF     LCD_CNTL, E     ; toggle E for LCD
                        0384             BCF     LCD_CNTL, E
                        0385             RETURN
                        0386 ;
                        0387     endif
                        0388 ;
                        0390 ;
                        0391     if ( Four_bit )     ; 4-bit Data transfers?
                        0392 ;
                        0393         if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
                        0394 ;
                        0395 ;*****************************************************************
                        0396 ;* This routine checks the busy flag, returns when not busy      *
                        0397 ;*  Affects:                                                      *
                        0398 ;*      TEMP - Returned with busy/address                        *
                        0399 ;*****************************************************************
                        0400 ;
                        0401 BUSY_CHECK
0083 1683               0402             BSF     STATUS, RP0     ; Select Register page 1
0084 30FF               0403             MOVLW   0xFF            ; Set Port_D for input
0085 0088               0404             MOVWF   LCD_DATA_TRIS
0086 1283               0405             BCF     STATUS, RP0     ; Select Register page 0
0087 1085               0406             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0088 1505               0407             BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0089 1585               0408             BSF     LCD_CNTL, E     ; Set E high
008A 1185               0409             BCF     LCD_CNTL, E     ; Set E low
008B 0808               0410             MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
008C 39F0               0411             ANDLW   0xF0            ; Mask out lower nibble
008D 00B5               0412             MOVWF   TEMP
008E 1585               0413             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
008F 1185               0414             BCF     LCD_CNTL, E
0090 0E08               0415             SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
0091 390F               0416             ANDLW   0x0F            ; Mask out upper nibble
0092 04B5               0417             IORWF   TEMP            ; Combine nibbles
0093 1BB5               0418             BTFSC   TEMP, 7         ; Check busy flag, high = busy
0094 2883               0419             GOTO    BUSY_CHECK      ; If busy, check again
0095 1105               0420             BCF     LCD_CNTL, R_W
0096 1683               0421             BSF     STATUS, RP0     ; Select Register page 1
0097 300F               0422             MOVLW   0x0F
```


© 1994 Microchip Technology Inc.

```
0098 0088          0423             MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
0099 1283          0424             BCF     STATUS, RP0     ; Select Register page 0
009A 0008          0425             RETURN
                   0426 ;
                   0427         else            ; 4-bit transfers on the low nibble of the PORT
                   0428 ;
                   0429 ;*****************************************************************
                   0430 ;* This routine checks the busy flag, returns when not busy      *
                   0431 ;*  Affects:                                                      *
                   0432 ;*       TEMP - Returned with busy/address                       *
                   0433 ;*****************************************************************
                   0434 ;
                   0435 BUSY_CHECK
                   0436             BSF     STATUS, RP0     ; Bank 1
                   0437             MOVLW   0xFF            ; Set PortB for input
                   0438             MOVWF   LCD_DATA_TRIS
                   0439             BCF     STATUS, RP0     ; Bank 0
                   0440             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
                   0441             BSF     LCD_CNTL, R_W   ; Setup to read busy flag
                   0442             BSF     LCD_CNTL, E     ; Set E high
                   0443             BCF     LCD_CNTL, E     ; Set E low
                   0444             SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
                   0445             ANDLW   0xF0            ; Mask out lower nibble
                   0446             MOVWF   TEMP            ;
                   0447             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
                   0448             BCF     LCD_CNTL, E
                   0449             MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
                   0450             ANDLW   0x0F            ; Mask out upper nibble
                   0451             IORWF   TEMP, F         ; Combine nibbles
                   0452             BTFSC   TEMP, 7         ; Check busy flag, high = busy
                   0453             GOTO    BUSY_CHECK      ; If busy, check again
                   0454             BCF     LCD_CNTL, R_W
                   0455             BSF     STATUS, RP0     ; Bank 1
                   0456             MOVLW   0xF0            ;
                   0457             MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
                   0458             BCF     STATUS, RP0     ; Bank 0
                   0459             RETURN
                   0460 ;
                   0461         endif
                   0462     else
                   0463 ;
                   0464 ;*************************************************************
                   0465 ;* This routine checks the busy flag, returns when not busy  *
                   0466 ;*  Affects:                                                  *
                   0467 ;*       TEMP - Returned with busy/address                   *
                   0468 ;*************************************************************
                   0469 ;
                   0470 BUSY_CHECK
```

© 1994 Microchip Technology Inc.

```
                    0471            BSF     STATUS,RP0      ; Select Register page 1
                    0472            MOVLW   0xFF            ; Set port_D for input
                    0473            MOVWF   LCD_DATA_TRIS
                    0474            BCF     STATUS, RP0     ; Select Register page 0
                    0475            BCF     LCD_CNTL, RS    ; Set LCD for command mode
                    0476            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
                    0477            BSF     LCD_CNTL, E     ; Set E high
                    0478            BCF     LCD_CNTL, E     ; Set E low
                    0479            MOVF    LCD_DATA, w     ; Read busy flag, DDram address
                    0480            MOVWF   TEMP
                    0481            BTFSC   TEMP, 7         ; Check busy flag, high=busy
                    0482            GOTO    BUSY_CHECK
                    0483            BCF     LCD_CNTL, R_W
                    0484            BSF     STATUS, RP0     ; Select Register page 1
                    0485            MOVLW   0x00
                    0486            MOVWF   LCD_DATA_TRIS   ; Set port_D for output
                    0487            BCF     STATUS, RP0     ; Select Register page 0
                    0488            RETURN
                    0489 ;
                    0490    endif
                    0492 ;
                    0493 Table
009B 0782           0494            addwf   PCL             ;Jump to char pointed to in W reg
009C 344D           0495            retlw   'M'
009D 3469           0496            retlw   'i'
009E 3463           0497            retlw   'c'
009F 3472           0498            retlw   'r'
00A0 346F           0499            retlw   'o'
00A1 3463           0500            retlw   'c'
00A2 3468           0501            retlw   'h'
00A3 3469           0502            retlw   'i'
00A4 3470           0503            retlw   'p'
00A5 3420           0504            retlw   ' '
00A6 3454           0505            retlw   'T'
00A7 3465           0506            retlw   'e'
00A8 3463           0507            retlw   'c'
00A9 3468           0508            retlw   'h'
00AA 346E           0509            retlw   'n'
00AB 346F           0510            retlw   'o'
00AC 346C           0511            retlw   'l'
00AD 346F           0512            retlw   'o'
00AE 3467           0513            retlw   'g'
00AF 3479           0514            retlw   'y'
                    0515 Table_End
00B0 3400           0516            retlw   0
                    0517 ;
                    0518        if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                    0519       MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
```


© 1994 Microchip Technology Inc.

```
                    0520     endif
                    0521 ;
                    0522
                    0523
                    0524
                    0525     end
                    0526
                    0527
                    0528
                    0529
                    0530


MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:59:12                PAGE 14


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X-------
00C0 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :    13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

© 1994 Microchip Technology Inc.

APPENDIX D: 4-BIT DATA INTERFACE, LOW NIBBLE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  5:29:26                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;
                     0012 ;
                     0013        include <C74_reg.h>
                     0233
                     0013
                     0014        include <lm032l.h>
                     0014
                     0015 ;
0001                 0016 Four_bit          EQU     TRUE       ; Selects 4- or 8-bit data transfers
0000                 0017 Data_HI           EQU     FALSE       ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;
                     0019 ;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;
0006                 0022 LCD_DATA          EQU     PORTB
0086                 0023 LCD_DATA_TRIS     EQU     TRISB
                     0024 ;
                     0025     else
                     0026 ;
                     0027 LCD_DATA          EQU     PORTD
                     0028 LCD_DATA_TRIS     EQU     TRISD
                     0029 ;
                     0030     endif
                     0031 ;
0005                 0032 LCD_CNTL          EQU     PORTA
                     0033 ;
                     0034 ;
                     0035 ;
```


© 1994 Microchip Technology Inc.

```
                        0036 ; LCD Display Commands and Control Signal names.
                        0037 ;
                        0038     if ( Four_bit && !Data_HI )
                        0039 ;
0000                    0040 E                          EQU             0                       ; LCD Enable control line
0001                    0041 R_W                        EQU             1                       ; LCD Read/Write control line
0002                    0042 RS                         EQU             2                       ; LCD Register Select control
line
                        0043 ;
                        0044     else
                        0045 ;
                        0046 E                          EQU             3                       ; LCD Enable control line
                        0047 R_W                        EQU             2                       ; LCD Read/Write control line
                        0048 RS                         EQU             1                       ; LCD Register Select control
line
                        0049 ;
                        0050     endif
                        0051 ;
                        0052 ;
0030                    0053 TEMP1      EQU     0x030
                        0054 ;
                        0055         org     RESET_V                 ; RESET vector location
0000 2808               0056 RESET      GOTO    START                   ;
                        0057 ;
                        0058 ; This is the Periperal Interrupt routine. Should NOT get here
                        0059 ;
                        0061         org     ISR_V              ; Interrupt vector location
                        0062 PER_INT_V
0004 1283               0063 ERROR1     BCF     STATUS, RP0         ; Bank 0
0005 1407               0064            BSF     PORTC, 0
0006 1007               0065            BCF     PORTC, 0
0007 2804               0066            GOTO    ERROR1
                        0067 ;
                        0068 ;
                        0069 ;
                        0070 START                              ; POWER_ON Reset (Beginning of program)
0008 0183               0071            CLRF    STATUS          ; Do initialization (Bank 0)
0009 018B               0072            CLRF    INTCON
000A 018C               0073            CLRF    PIR1
000B 1683               0074            BSF     STATUS, RP0     ; Bank 1
000C 3000               0075            MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
000D 0081               0076            MOVWF   OPTION_R        ;
000E 018C               0077            CLRF    PIE1            ; Disable all peripheral interrupts
000F 30FF               0078            MOVLW   0xFF            ;
0010 009F               0079            MOVWF   ADCON1          ; Port A is Digital.
                        0080 ;
                        0081 ;
0011 1283               0082            BCF     STATUS, RP0     ; Bank 0
```

© 1994 Microchip Technology Inc.

```
0012 0185          0083             CLRF    PORTA           ; ALL PORT output should output Low.
0013 0186          0084             CLRF    PORTB
0014 0187          0085             CLRF    PORTC
0015 0188          0086             CLRF    PORTD
0016 0189          0087             CLRF    PORTE
0017 1010          0088             BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                   0089 ;
0018 1683          0090             BSF     STATUS, RP0     ; Select Bank 1
0019 0185          0091             CLRF    TRISA           ; RA5 - 0 outputs
001A 30F0          0092             MOVLW   0xF0            ;
001B 0086          0093             MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187          0094             CLRF    TRISC           ; RC Port are outputs
001D 1407          0095             BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188          0096             CLRF    TRISD           ; RD Port are outputs
001F 0189          0097             CLRF    TRISE           ; RE Port are outputs
0020 140C          0098             BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781          0099             BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283          0100             BCF     STATUS, RP0     ; Select Bank 0
                   0101 ;
                   0103 ;
                   0104 ; Initilize the LCD Display Module
                   0105 ;
0023 0185          0106             CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                   0107
                   0108 DISPLAY_INIT
                   0109     if ( Four_bit && !Data_HI )
0024 3002          0110             MOVLW   0x02            ; Command for 4-bit interface low nibble
                   0111     endif
                   0112 ;
                   0113     if ( Four_bit && Data_HI )
                   0114             MOVLW   0x020           ; Command for 4-bit interface high nibble
                   0115     endif
                   0116 ;
                   0117     if ( !Four_bit )
                   0118             MOVLW   0x038           ; Command for 8-bit interface
                   0119     endif
                   0120 ;
0025 0086          0121             MOVWF   LCD_DATA        ;
0026 1405          0122             BSF     LCD_CNTL, E     ;
0027 1005          0123             BCF     LCD_CNTL, E     ;
                   0124 ;
                   0125 ; This routine takes the calculated times that the delay loop needs to
                   0126 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                   0127 ; frequency of operation. These uses registers before they are needed to
                   0128 ; store the time.
                   0129 ;
0028 3006          0130 LCD_DELAY   MOVLW   LCD_INIT_DELAY  ;
0029 00B3          0131             MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
```


© 1994 Microchip Technology Inc.

```
002A 01B4          0132              CLRF    LSD             ;
002B 0BB4          0133 LOOP2        DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002C 282B          0134              GOTO    LOOP2           ;
002D 0BB3          0135              DECFSZ  MSD             ;
                   0136 END_LCD_DELAY
002E 282B          0137              GOTO    LOOP2           ;
                   0138 ;
                   0139 ; Command sequence for 2 lines of 5x7 characters
                   0140 ;
                   0141 CMD_SEQ
                   0142 ;
                   0143     if ( Four_bit )
                   0144         if ( !Data_HI )
002F 3002          0145             MOVLW   0X02            ; 4-bit low nibble xfer
                   0146         else
                   0147             MOVLW   0X020           ; 4-bit high nibble xfer
                   0148         endif
                   0149 ;
                   0150     else                            ; 8-bit mode
                   0151             MOVLW   0X038
                   0152     endif
                   0153 ;
0030 0086          0154             MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
0031 1405          0155             BSF     LCD_CNTL, E     ;
0032 1005          0156             BCF     LCD_CNTL, E     ;
                   0157 ;
                   0158     if ( Four_bit )                 ; This code for only 4-bit mode (2nd xfer)
                   0159         if ( !Data_HI )
0033 3008          0160             MOVLW   0x08            ; 4-bit low nibble xfer
                   0161         else
                   0162             MOVLW   0x080           ; 4-bit high nibble xfer
                   0163         endif
0034 0086          0164             MOVWF   LCD_DATA        ;
0035 1405          0165             BSF     LCD_CNTL, E     ;
0036 1005          0166             BCF     LCD_CNTL, E     ;
                   0167     endif
                   0168 ;
                   0169 ; Busy Flag should be valid after this point
                   0170 ;
0037 300C          0171             MOVLW   DISP_ON         ;
0038 2074          0172             CALL    SEND_CMD        ;
0039 3001          0173             MOVLW   CLR_DISP        ;
003A 2074          0174             CALL    SEND_CMD        ;
003B 3006          0175             MOVLW   ENTRY_INC       ;
003C 2074          0176             CALL    SEND_CMD        ;
003D 3080          0177             MOVLW   DD_RAM_ADDR     ;
003E 2074          0178             CALL    SEND_CMD        ;
                   0179 ;
```

© 1994 Microchip Technology Inc.

```
                        0181 ;
                        0182 ;Send a message the hard way
003F 304D               0183            movlw   'M'
0040 2065               0184            call    SEND_CHAR
0041 3069               0185            movlw   'i'
0042 2065               0186            call    SEND_CHAR
0043 3063               0187            movlw   'c'
0044 2065               0188            call    SEND_CHAR
0045 3072               0189            movlw   'r'
0046 2065               0190            call    SEND_CHAR
0047 306F               0191            movlw   'o'
0048 2065               0192            call    SEND_CHAR
0049 3063               0193            movlw   'c'
004A 2065               0194            call    SEND_CHAR
004B 3068               0195            movlw   'h'
004C 2065               0196            call    SEND_CHAR
004D 3069               0197            movlw   'i'
004E 2065               0198            call    SEND_CHAR
004F 3070               0199            movlw   'p'
0050 2065               0200            call    SEND_CHAR
                        0201
0051 30C0               0202            movlw   B'11000000'     ;Address DDRam first character, second line
0052 2074               0203            call    SEND_CMD
                        0204
                        0205 ;Demonstration of the use of a table to output a message
0053 3000               0206            movlw   0               ;Table address of start of message
                        0207 dispmsg
0054 00B0               0208            movwf   TEMP1           ;TEMP1 holds start of message address
0055 209B               0209            call    Table
0056 39FF               0210            andlw   0FFh            ;Check if at end of message (zero
0057 1903               0211            btfsc   STATUS,Z        ;returned at end)
0058 285D               0212            goto    out
0059 2065               0213            call    SEND_CHAR       ;Display character
005A 0830               0214            movf    TEMP1,w         ;Point to next character
005B 3E01               0215            addlw   1
005C 2854               0216            goto    dispmsg
                        0217 out
                        0218 loop
005D 285D               0219            goto    loop            ;Stay here forever
                        0220 ;
                        0221 ;
                        0222 INIT_DISPLAY
005E 300C               0223            MOVLW   DISP_ON             ; Display On, Cursor On
005F 2074               0224            CALL    SEND_CMD            ; Send This command to the Display Module
0060 3001               0225            MOVLW   CLR_DISP            ; Clear the Display
0061 2074               0226            CALL    SEND_CMD            ; Send This command to the Display Module
0062 3006               0227            MOVLW   ENTRY_INC           ; Set Entry Mode Inc., No shift
0063 2074               0228            CALL    SEND_CMD            ; Send This command to the Display Module
```


© 1994 Microchip Technology Inc.

```
0064 0008          0229              RETURN
                   0230 ;
                   0232 ;*****************************************************************
                   0233 ;* The LCD Module Subroutines                                    *
                   0234 ;*****************************************************************
                   0235 ;
                   0236     if ( Four_bit )       ; 4-bit Data transfers?
                   0237 ;
                   0238         if ( Data_HI )    ; 4-bit transfers on the high nibble of the PORT
                   0239 ;
                   0240 ;*****************************************************************
                   0241 ;*SendChar - Sends character to LCD                            *
                   0242 ;*This routine splits the character into the upper and lower   *
                   0243 ;*nibbles and sends them to the LCD, upper nibble first.       *
                   0244 ;*****************************************************************
                   0245 ;
                   0246 SEND_CHAR
                   0247              MOVWF   CHAR            ;Character to be sent is in W
                   0248              CALL    BUSY_CHECK      ;Wait for LCD to be ready
                   0249              MOVF    CHAR, w
                   0250              ANDLW   0xF0            ;Get upper nibble
                   0251              MOVWF   LCD_DATA        ;Send data to LCD
                   0252              BCF     LCD_CNTL, R_W   ;Set LCD to read
                   0253              BSF     LCD_CNTL, RS    ;Set LCD to data mode
                   0254              BSF     LCD_CNTL, E     ;toggle E for LCD
                   0255              BCF     LCD_CNTL, E
                   0256              SWAPF   CHAR, w
                   0257              ANDLW   0xF0            ;Get lower nibble
                   0258              MOVWF   LCD_DATA        ;Send data to LCD
                   0259              BSF     LCD_CNTL, E     ;toggle E for LCD
                   0260              BCF     LCD_CNTL, E
                   0261              RETURN
                   0262 ;
                   0263          else              ; 4-bit transfers on the low nibble of the PORT
                   0264 ;
                   0265 ;*****************************************************************
                   0266 ;* SEND_CHAR - Sends character to LCD                          *
                   0267 ;* This routine splits the character into the upper and lower   *
                   0268 ;* nibbles and sends them to the LCD, upper nibble first.       *
                   0269 ;* The data is transmitted on the PORT<3:0> pins                *
                   0270 ;*****************************************************************
                   0271 ;
                   0272 SEND_CHAR
0065 00B6          0273              MOVWF   CHAR            ; Character to be sent is in W
0066 2083          0274              CALL    BUSY_CHECK      ; Wait for LCD to be ready
0067 0E36          0275              SWAPF   CHAR, W
0068 390F          0276              ANDLW   0x0F            ; Get upper nibble
```

© 1994 Microchip Technology Inc.

```
0069 0086           0278            MOVWF   LCD_DATA        ; Send data to LCD
006A 1085           0279            BCF     LCD_CNTL, R_W   ; Set LCD to read
006B 1505           0280            BSF     LCD_CNTL, RS    ; Set LCD to data mode
006C 1405           0281            BSF     LCD_CNTL, E     ; toggle E for LCD
006D 1005           0282            BCF     LCD_CNTL, E
006E 0836           0283            MOVF    CHAR, W
006F 390F           0284            ANDLW   0x0F            ; Get lower nibble
0070 0086           0285            MOVWF   LCD_DATA        ; Send data to LCD
0071 1405           0286            BSF     LCD_CNTL, E     ; toggle E for LCD
0072 1005           0287            BCF     LCD_CNTL, E
0073 0008           0288            RETURN
                    0289 ;
                    0290        endif
                    0291    else
                    0292 ;
                    0293 ;*****************************************************************
                    0294 ;* SEND_CHAR - Sends character contained in register W to LCD    *
                    0295 ;* This routine sends the entire character to the PORT           *
                    0296 ;* The data is transmitted on the PORT<7:0> pins                 *
                    0297 ;*****************************************************************
                    0298 ;
                    0299 SEND_CHAR
                    0300            MOVWF   CHAR            ; Character to be sent is in W
                    0301            CALL    BUSY_CHECK      ; Wait for LCD to be ready
                    0302            MOVF    CHAR, w
                    0303            MOVWF   LCD_DATA        ; Send data to LCD
                    0304            BCF     LCD_CNTL, R_W   ; Set LCD in read mode
                    0305            BSF     LCD_CNTL, RS    ; Set LCD in data mode
                    0306            BSF     LCD_CNTL, E     ; toggle E for LCD
                    0307            BCF     LCD_CNTL, E
                    0308            RETURN
                    0309 ;
                    0310    endif
                    0311 ;
                    0313 ;
                    0314 ;*****************************************************************
                    0315 ;* SendCmd - Sends command to LCD                               *
                    0316 ;* This routine splits the command into the upper and lower      *
                    0317 ;* nibbles and sends them to the LCD, upper nibble first.        *
                    0318 ;* The data is transmitted on the PORT<3:0> pins                 *
                    0319 ;*****************************************************************
                    0320 ;
                    0321    if ( Four_bit )      ; 4-bit Data transfers?
                    0322 ;
                    0323        if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
                    0324 ;
                    0325 ;*****************************************************************
```


© 1994 Microchip Technology Inc.

```
                0326 ;* SEND_CMD - Sends command to LCD                                *
                0327 ;* This routine splits the command into the upper and lower       *
                0328 ;* nibbles and sends them to the LCD, upper nibble first.         *
                0329 ;*********************************************************************
                0330
                0331 SEND_CMD
                0332             MOVWF   CHAR            ; Character to be sent is in W
                0333             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                0334             MOVF    CHAR,w
                0335             ANDLW   0xF0            ; Get upper nibble
                0336             MOVWF   LCD_DATA        ; Send data to LCD
                0337             BCF     LCD_CNTL,R_W    ; Set LCD to read
                0338             BCF     LCD_CNTL,RS     ; Set LCD to command mode
                0339             BSF     LCD_CNTL,E      ; toggle E for LCD
                0340             BCF     LCD_CNTL,E
                0341             SWAPF   CHAR,w
                0342             ANDLW   0xF0            ; Get lower nibble
                0343             MOVWF   LCD_DATA        ; Send data to LCD
                0344             BSF     LCD_CNTL,E      ; toggle E for LCD
                0345             BCF     LCD_CNTL,E
                0346             RETURN
                0347 ;
                0348         else            ; 4-bit transfers on the low nibble of the PORT
                0349 ;
                0350 SEND_CMD
0074 00B6       0351             MOVWF   CHAR            ; Character to be sent is in W
0075 2083       0352             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0076 0E36       0353             SWAPF   CHAR, W
0077 390F       0354             ANDLW   0x0F            ; Get upper nibble
0078 0086       0355             MOVWF   LCD_DATA        ; Send data to LCD
0079 1085       0356             BCF     LCD_CNTL, R_W   ; Set LCD to read
007A 1105       0357             BCF     LCD_CNTL, RS    ; Set LCD to command mode
007B 1405       0358             BSF     LCD_CNTL, E     ; toggle E for LCD
007C 1005       0359             BCF     LCD_CNTL, E
007D 0836       0360             MOVF    CHAR, W
007E 390F       0361             ANDLW   0x0F            ; Get lower nibble
007F 0086       0362             MOVWF   LCD_DATA        ; Send data to LCD
0080 1405       0363             BSF     LCD_CNTL, E     ; toggle E for LCD
0081 1005       0364             BCF     LCD_CNTL, E
0082 0008       0365             RETURN
                0366 ;
                0367         endif
                0368     else
                0369 ;
                0370 ;*************************************************************
                0371 ;* SEND_CND - Sends command contained in register W to LCD   *
                0372 ;* This routine sends the entire character to the PORT       *
                0373 ;* The data is transmitted on the PORT<7:0> pins             *
```

© 1994 Microchip Technology Inc.

```
0374 ;*****************************************************************
0375
0376 SEND_CMD
0377             MOVWF   CHAR            ; Command to be sent is in W
0378             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0379             MOVF    CHAR, w
0380             MOVWF   LCD_DATA        ; Send data to LCD
0381             BCF     LCD_CNTL, R_W   ; Set LCD in read mode
0382             BCF     LCD_CNTL, RS    ; Set LCD in command mode
0383             BSF     LCD_CNTL, E     ; toggle E for LCD
0384             BCF     LCD_CNTL, E
0385             RETURN
0386 ;
0387     endif
0388 ;
0390 ;
0391     if ( Four_bit )     ; 4-bit Data transfers?
0392 ;
0393         if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
0394 ;
0395 ;*****************************************************************
0396 ;* This routine checks the busy flag, returns when not busy      *
0397 ;*  Affects:                                                     *
0398 ;*      TEMP - Returned with busy/address                        *
0399 ;*****************************************************************
0400 ;
0401 BUSY_CHECK
0402             BSF     STATUS, RP0     ; Select Register page 1
0403             MOVLW   0xFF            ; Set Port_D for input
0404             MOVWF   LCD_DATA_TRIS
0405             BCF     STATUS, RP0     ; Select Register page 0
0406             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0407             BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0408             BSF     LCD_CNTL, E     ; Set E high
0409             BCF     LCD_CNTL, E     ; Set E low
0410             MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0411             ANDLW   0xF0            ; Mask out lower nibble
0412             MOVWF   TEMP
0413             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0414             BCF     LCD_CNTL, E
0415             SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
0416             ANDLW   0x0F            ; Mask out upper nibble
0417             IORWF   TEMP            ; Combine nibbles
0418             BTFSC   TEMP, 7         ; Check busy flag, high = busy
0419             GOTO    BUSY_CHECK      ; If busy, check again
0420             BCF     LCD_CNTL, R_W
0421             BSF     STATUS, RP0     ; Select Register page 1
0422             MOVLW   0x0F
```


© 1994 Microchip Technology Inc.

```
                0423             MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
                0424             BCF     STATUS, RP0     ; Select Register page 0
                0425             RETURN
                0426 ;
                0427         else                ; 4-bit transfers on the low nibble of the PORT
                0428 ;
                0429 ;*****************************************************************
                0430 ;* This routine checks the busy flag, returns when not busy        *
                0431 ;*  Affects:                                                       *
                0432 ;*      TEMP - Returned with busy/address                          *
                0433 ;*****************************************************************
                0434 ;
                0435 BUSY_CHECK
 0083 1683      0436             BSF     STATUS, RP0     ; Bank 1
 0084 30FF      0437             MOVLW   0xFF            ; Set PortB for input
 0085 0086      0438             MOVWF   LCD_DATA_TRIS
 0086 1283      0439             BCF     STATUS, RP0     ; Bank 0
 0087 1105      0440             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
 0088 1485      0441             BSF     LCD_CNTL, R_W   ; Setup to read busy flag
 0089 1405      0442             BSF     LCD_CNTL, E     ; Set E high
 008A 1005      0443             BCF     LCD_CNTL, E     ; Set E low
 008B 0E06      0444             SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
 008C 39F0      0445             ANDLW   0xF0            ; Mask out lower nibble
 008D 00B5      0446             MOVWF   TEMP            ;
 008E 1405      0447             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
 008F 1005      0448             BCF     LCD_CNTL, E
 0090 0806      0449             MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
 0091 390F      0450             ANDLW   0x0F            ; Mask out upper nibble
 0092 04B5      0451             IORWF   TEMP, F         ; Combine nibbles
 0093 1BB5      0452             BTFSC   TEMP, 7         ; Check busy flag, high = busy
 0094 2883      0453             GOTO    BUSY_CHECK      ; If busy, check again
 0095 1085      0454             BCF     LCD_CNTL, R_W
 0096 1683      0455             BSF     STATUS, RP0     ; Bank 1
 0097 30F0      0456             MOVLW   0xF0            ;
 0098 0086      0457             MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
 0099 1283      0458             BCF     STATUS, RP0     ; Bank 0
 009A 0008      0459             RETURN
                0460 ;
                0461         endif
                0462     else
                0463 ;
                0464 ;*****************************************************************
                0465 ;* This routine checks the busy flag, returns when not busy   *
                0466 ;*  Affects:                                                   *
                0467 ;*      TEMP - Returned with busy/address                      *
                0468 ;*****************************************************************
                0469 ;
                0470 BUSY_CHECK
```

© 1994 Microchip Technology Inc.

```
                    0471            BSF     STATUS,RP0      ; Select Register page 1
                    0472            MOVLW   0xFF            ; Set port_D for input
                    0473            MOVWF   LCD_DATA_TRIS
                    0474            BCF     STATUS, RP0     ; Select Register page 0
                    0475            BCF     LCD_CNTL, RS    ; Set LCD for command mode
                    0476            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
                    0477            BSF     LCD_CNTL, E     ; Set E high
                    0478            BCF     LCD_CNTL, E     ; Set E low
                    0479            MOVF    LCD_DATA, w     ; Read busy flag, DDram address
                    0480            MOVWF   TEMP
                    0481            BTFSC   TEMP, 7         ; Check busy flag, high=busy
                    0482            GOTO    BUSY_CHECK
                    0483            BCF     LCD_CNTL, R_W
                    0484            BSF     STATUS, RP0     ; Select Register page 1
                    0485            MOVLW   0x00
                    0486            MOVWF   LCD_DATA_TRIS   ; Set port_D for output
                    0487            BCF     STATUS, RP0     ; Select Register page 0
                    0488            RETURN
                    0489 ;
                    0490     endif
                    0492 ;
                    0493 Table
009B 0782           0494            addwf   PCL             ;Jump to char pointed to in W reg
009C 344D           0495            retlw   'M'
009D 3469           0496            retlw   'i'
009E 3463           0497            retlw   'c'
009F 3472           0498            retlw   'r'
00A0 346F           0499            retlw   'o'
00A1 3463           0500            retlw   'c'
00A2 3468           0501            retlw   'h'
00A3 3469           0502            retlw   'i'
00A4 3470           0503            retlw   'p'
00A5 3420           0504            retlw   ' '
00A6 3454           0505            retlw   'T'
00A7 3465           0506            retlw   'e'
00A8 3463           0507            retlw   'c'
00A9 3468           0508            retlw   'h'
00AA 346E           0509            retlw   'n'
00AB 346F           0510            retlw   'o'
00AC 346C           0511            retlw   'l'
00AD 346F           0512            retlw   'o'
00AE 3467           0513            retlw   'g'
00AF 3479           0514            retlw   'y'
                    0515 Table_End
00B0 3400           0516            retlw   0
                    0517 ;
                    0518        if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                    0519       MESSG    "Warning - User Definded: Table Table crosses page boundry in computed jump"
```


© 1994 Microchip Technology Inc.

```
                        0520     endif
                        0521 ;
                        0522
                        0523
                        0524
                        0525     end
                        0526
                        0527
                        0528
                        0529
                        0530
```

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  5:29:26                PAGE 14


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X---------------
00C0 : ---------------- ---------------- ---------------- ----------------

All other memory blocks unused.

Errors   :     0
Warnings :    13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

© 1994 Microchip Technology Inc.

APPENDIX E: LM032L.H

```
                    nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;                   PROGRAM:        CLOCK.H
;                   Revision:       5-10-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq            EQU             D'4000000'      ; Device Frequency is 4 MHz
DB_HI_BYTE          EQU             (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY      EQU             (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR          EQU             40              ; RAM Location
OUTER_CNTR          EQU             41              ; RAM Location

T1OSO               EQU             0               ; The RC0 / T1OSO / T1CKI
;
RESET_V             EQU             0x0000          ; Address of RESET Vector
ISR_V               EQU             0x0004          ; Address of Interrupt Vector
PMEM_END            EQU             0x07FF          ; Last address in Program Memory
TABLE_ADDR          EQU             0x0400          ; Address where to start Tables
;
HR_MIN_SW           EQU             0x7             ; The switch to select the units
INC_SW              EQU             0x6             ; The switch to increment the selected units
CLR_MIN_SW          EQU             0x5             ; The switch to clear the minutes and seconds
;
FLAG_REG            EQU             0x020           ; Register which contains flag bits
;
;  +-------+-----+-----+-----------+-----+-----+----------+----------+
;  |  AM   |  -  |  -  | KEY_INPUT |  -  |  -  | MIN_UNIT | HR_UNIT  |
;  +-------+-----+-----+-----------+-----+-----+----------+----------+
;
AM                  EQU             0x07            ; Flag to specify if AM or PM
;
KEY_INPUT           EQU             0x04            ; Flag to specify if doing key inputs
;
MIN_UNIT            EQU             0x01            ; Flags to specify which units to operate on
HR_UNIT             EQU             0x00            ;  (HRS, MIN, or none)
;
HRS                 EQU             0x030           ; Holds counter value for HOURS
MIN                 EQU             0x031           ; Holds counter value for MINUTES
SECS                EQU             0x032           ; Holds counter value for SECONDS
```


© 1994 Microchip Technology Inc.

```
MSD             EQU     0x033   ; Temporary register, Holds Most Significant Digit of BIN to BCD conversion
LSD             EQU     0x034   ; Temporary register, Holds Least Significant Digit of BIN to BCD conversion
TEMP            EQU     0x035   ; Temporary register
CHAR            EQU     0x036   ; Temporary register, Holds value to send to LCD module.
;
WAIT_CNTR       EQU     0x040   ; Counter that holds wait time for key inputs
;
;
; LCD Module commands
;
DISP_ON         EQU     0x00C   ; Display on
DISP_ON_C       EQU     0x00E   ; Display on, Cursor on
DISP_ON_B       EQU     0x00F   ; Display on, Cursor on, Blink cursor
DISP_OFF        EQU     0x008   ; Display off
CLR_DISP        EQU     0x001   ; Clear the Display
ENTRY_INC       EQU     0x006   ;
ENTRY_INC_S     EQU     0x007   ;
ENTRY_DEC       EQU     0x004   ;
ENTRY_DEC_S     EQU     0x005   ;
DD_RAM_ADDR     EQU     0x080   ; Least Significant 7-bit are for address
DD_RAM_UL       EQU     0x080   ; Upper Left coner of the Display
;

                list
```

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Using Timer1 in Asynchronous Clock Mode

INTRODUCTION

This application note discusses the use of the Timer1 module, of the PIC16CXX family, for an asynchronous clock. The Timer1 module has it own oscilator circuitry, which allows the timer to keep real time, even when the device is in sleep mode. When the device is in sleep, the oscillator will continue to increment Timer1. An overflow of Timer1 causes a timer1 interrupt (if enabled) and will wake the processor from sleep. The interrupt service routine, then can do the desired task.

OVERVIEW

Timer1 is a 16-bit counter with a 2-bit prescaler. Timer1 can be incremented from either the internal clock, an external clock, or an external oscillator. Timer1 can be configured to synchronize or not synchronize the external clock sources. The asynchronous operation allows timer1 to increment when the device is in sleep. Figure 1 is a block diagram of Timer1.

To set up Timer1 for asynchronous operation the timer1 control register (T1CON) must have the following bits configured:

- TMR1CS set (external clock source)
- T1CKS<1:0> configured for the desired prescaler
- T1INSYNC set (asynchronous operation)
- TMR1ON set (enables Timer1)
- Set T1OSCEN, if using an external oscilator

In asynchronous operation, if the clock source is an external clock, the clock must be on the T1CKI pin. If the clock source is a crystal oscillator, the crystal is connected accross the T1OSO and T1OSI pins. Please refer to the device Data Sheet for recommended capacitor selection for the Timer1 oscillator.

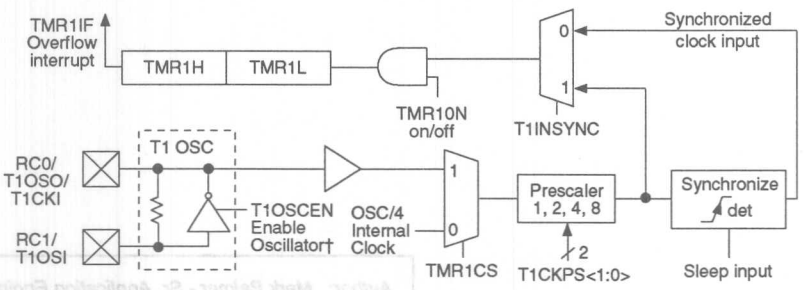
When using Timer1 in asynchronous mode, the use of an external clock minimizes the operating and sleep currents. This is because the oscillator circuitry is disabled. Though the external clock may give the lower device currents, the use of a crystal oscillator may lead to lower system currant consumption and system cost.

System current consumption can also be reduced by having the timer1 overflow interrupt wake the processor from sleep at the desired interval, With a 32.768 KHz crystal, Timer1's overflow rate ranges from 2 to 16 seconds, depending on the prescaler chosen. Table 1 shows timer1 overflow times for various crystal frequencies and prescaler values.

TABLE 1: TIMER1 OVERFLOW TIMES

Prescaler	Frequency (KHz)		
	32.768	100	200
1	2 Seconds	0.655	0.327 Seconds
2	4 Seconds	1.31 Seconds	0.655 Seconds
4	8 Seconds	2.62 Seconds	1.31 Seconds
8	16 Seconds	5.24 Seconds	2.62 Seconds

FIGURE 1: TIMER1 BLOCK DIAGRAM



† When the T1 OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

© 1994 Microchip Technology Inc.

As can be seen the 32 Khz crystal, gives very nice overflow rates. These crystals, refered to as watch crystals, also can be relativly inexpensive. In many applications the 2 second overflow time, of a 32 KHz crystal, is too long. An easy way to reduce the the overflow time is during the interrupt service routine, to load the TMR1H register with a value. Table 2 shows the overflow times, depending on the value loaded into the TMR1H register and a prescaler of 1.

TABLE 2: TMR1H LOAD VALUES / TIMER1 OVERFLOW TIMES

TMR1H Load Value	Overflow Time (@ 32.768 KHz)
80 h	1 Second
C0 h	0.5 Seconds
E0 h	0.25 Seconds
F0 h	0.125 Seconds

Note: The loading of either TMR1H or TMR1L causes the prescaler to be cleared. When Timer1 is in operation, extreme care should be taken in modifying either the TMR1H or TMR1L registers, since this automatically modifies the prescaler to 1.

The code segment shown in Example 1 initializes the Timer1 module for asynchronous mode, enables the Timer1 interrupt, and the interrupt service routine loads the TMR1H register with a value.

CONCLUSION

Timer1 gives designers a powerful timebase function. The asynchronous operation and oscillator circuitry gives designers the ability to easily keep real time, while minimizing power consumption and external logic.

Author: Mark Palmer - Sr. Application Engineer
Logic Products Division

© 1994 Microchip Technology Inc.

EXAMPLE 1: TIMER1 CODE SEGMENT FOR ASYNCHRONOUS OPERATION

```
      org   0x000
Reset_V     GOTO    START
;
      org   0x004
PER_INT_V
            BCF     STATUS, RP0     ; Bank 0
            BTFSC   PIR1, TMR1IF    ; Timer 1 overflowed?
            GOTO    T1_OVRFL        ; YES, Service the Timer1 Overflow Interrupt
;
; Should NEVER get here
;
ERROR1                              ; NO, Unknown Interrupt Source
            BSF     PORTD, 1        ; Toggle a port pin to indicate error
            BCF     PORTD, 1
            GOTO    ERROR1
;
T1_OVRFL
            BCF     PIR1, TMR1IF    ; Clear Timer 1 Interrupt Flag
            MOVLW   0x80            ; Since doing key inputs, clear TMR1
            MOVWF   TMR1H           ;   for 1 sec overflow.
            :
            :                       ; Do Interrupt stuff here
            :
            RETFIE                  ; Return / Enable Global Interrupts
;
;
START                               ; POWER_ON Reset (Beginning of program)
            CLRF    STATUS          ; Do initialization (Bank 0)
            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
            :
            :                       ; Do Initialization stuff here
            :
            MOVLW   0x80            ; TIM1H:TMR1L = 0x8000 gives 1 second
            MOVWF   TMR1H           ;   overflow, at 32 KHz.
            CLRF    TMR1L           ;
;
            CLRF    INTCON
            CLRF    PIR1
            BSF     STATUS, RP0     ; Bank 1
            CLRF    PIE1            ; Disable all peripheral interrupts
;
    if ( C74_REV_A )                ; See PIC16C74 Errata
            BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
    endif
            BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
;
; Initialize the Special Function Registers (SFR) interrupts
;
            BCF     STATUS, RP0     ; Bank 0
            CLRF    PIR1            ;
            BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
            BSF     INTCON, GIE     ; Enable all Interrupts
;
            MOVLW   0x0E
            MOVWF   T1CON           ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
            BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
;
zzz         SLEEP
            GOTO    zzz             ; Sleep, wait for TMR1 interrupt
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Low-Power Real Time Clock

INTRODUCTION

This application note implements a low-power real time clock using the Timer1 module of the PIC16CXX family of processors. Timer1 can operate from its own crystal source, which allows the timer to increment while the device is in sleep mode. The device is placed into sleep to minimize the current consumption. Only the events that require processing will wake the device from sleep. These are a key input and a Timer1 overflow.

OPERATION

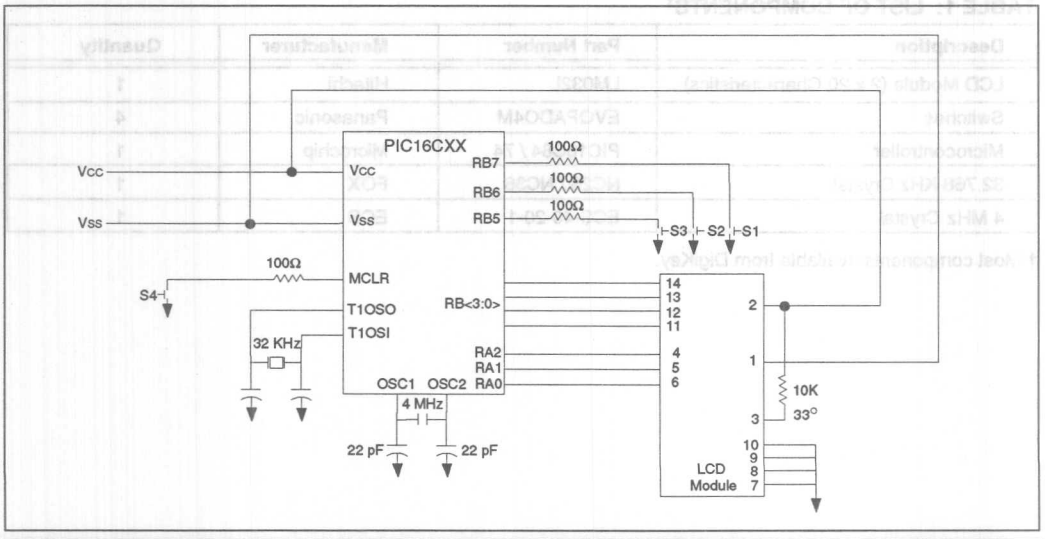
Upon power-up, the device goes into an initial state. This state sets the display to 12:00 PM and waits for Timer1 to generate an interrupt (every second). The Timer1 overflow interrupt wakes the device from sleep This causes the time registers (HRS, MIN, SECS) to be updated. If the SECS register contains an even value (SECS<0> = 0), the colon (":") is not displayed. This gives a visual indication for each second.

There are three keys for the setting of the clock. The SELECT_UNITS Key (S1) selects which units are to be modified (hours, minutes, off). The selected units are blanked for a second then flashed for one second. The INC Key (S2) increments the selected units. While incrementing, the selected units are displayed. After a key has not been depressed for more then one second, the selected units will begin to flash. The CLR_MIN Key (S3) clears the minutes and seconds. CLR_MIN is useful for exactly setting the time to the "top of the hour" as announced in radio broadcasts. After the INC or SELECT_UNITS keys are depressed, the user has ten seconds to depress the next key. After no key has been depressed for ten seconds, the unit returns to the clock mode.

To simplify the design time, a standard Hitachi LCD display module is used. In most applications requiring a LCD display, a custom LCD display is used. The LCD interface software would need to be modified to suit the specific LCD display driver being used.

Figure 1 is a block diagram of the design. RA<2:0> are the control signals to the LCD display, RB<3:0> is the 4-bit data bus, and RB<7:5> is the input switches. The OSC1 pin is connected to an RC network, which generates approximately a 4 MHz device frequency. The device frequency does not need to be stable, since the Timer1 module operates asynchronously. This allows the device's oscillaor to be configures for RC mode. This oscillator mode is the least expensive and has the quickest start-up time. Timer1 is where the accurate frequency is required. This crystal is connected to the T1OSI and T1OSO pins. A good choice for a crystal is a 32.786 KHz (watch) crystal. Table 1 is a list of the components and their part number.

FIGURE 1: CLOCK BLOCK DIAGRAM



© 1994 Microchip Technology Inc.

Relative to most microelectronics, the LCD's are slow devices. A good portion of the time spent in the Interrupt Service Routine, is talking to and updating the LCD module. To minimize power consumption, the device should be in the sleep mode as much as possible.

By using the conditional assemble, if a flag (called Debug) is true, the total time spent in the subroutine can be seen on the PORTD<0> pin (the high time). Measuring this time on an oscilloscope displayed a typical time of 800 uS that the device is awake. This 800 uS operation is out of the 1 second time that the device needs to service the interrupt (a TMR1 overflow).

The accuracy of a real time clock using Timer1 depends on the accuracy of the crystal being used. The more accurate the crystal, the higher the cost. So as always there is a cost / performance trade-off to be made. A crystal rated with an accuracy of 20 PPM (parts per million), could cause an error of about 1.7 seconds per day. For many applications, this should be adequate (said from someone who doesn't wear a watch).

The program presented here shows one method for a real time clock. Trade-offs between code size, current consumption and desired operation have been made. Some possible alternative implementations are:

1. When displaying the time, update only the characters that changed.
2. Turn off the display during sleep
3. LCD module data interface of 8-bits, not the 4-bit interface.

Alternative 1 can reduce the time awake, by keeping track of which characters need to be updated. The majority of the time it will be only the position which contains either the ":" or the " ". Next would be the ones place of the minutes, then the tens place of the minutes, etc. The display would only need to be completely updated 2 times every 24 hrs. This would reduce the amount of time talking with the LCD display at the cost of some program / data memory.

Depending on the requirements of the application and the characteristics of the display, alternative 2 could be implemented by turning the power off and on (at a given rate) to the display. This technique may lead to a lower system current consumption. Evaluation upon the desired display / display driver is recommended.

Alternative 3 uses the LCD module in an 8-bit mode, will reduce the size of the display routines (save about 20 words of program memory) at the cost of four additional I/O lines. For some applications this may be a good trade off to get the additional program memory space. The percentage of operating time saved is slight and should not give substantial power savings.

TABLE 1: LIST OF COMPONENTS†

Description	Part Number	Manufacturer	Quantity
LCD Module (2 x 20 Characteristics)	LM032L	Hitachi	1
Switches	EVQPADO4M	Panasonic	4
Microcontroller	PIC16C64 / 74	Microchip	1
32.768 KHz Crystal	NC26 / NC38	FOX	1
4 MHz Crystal	ECS-40-20-1	ECS	1

† Most components available from DigiKey.

© 1994 Microchip Technology Inc.

CONCLUSION

The Timer1 module allows many applications to include a real time clock at minimal system cost. This time function can be useful in consumer applications (display time) as well as in industrial applications (data time stamp). The accuracy of the time is strictly dependent on the accuracy of the crystal. Table 2 shows the program resource requirements.

TABLE 2: PROGRAM RESOURCE REQUIREMENTS

Resource			Words / Bytes	Cycles
Program Memory	Initialization		61	61
	Clock Operation	Increment Time W.C.	106	35 + Display
		Key Input W.C.		35 + Display Time
Data Memory	Display‡		208	526†
	Variables		5	N.A.
	Scratch RAM		4	N.A.

† Dependent on LCD Module (re: BUSY_CHECK subroutine)

‡ Assumes worst case numbers and best case response from LCD module.

Author: Mark Palmer - Sr. Application Engineer
Logic Products Division

© 1994 Microchip Technology Inc.

APPENDIX A: SOURCE CODE LISTING (CLOCK_01.LST)

```
MPASM 01.01 Released     CLOCK.ASM   5-13-1994  13:11:9              PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program implements a real time clock using the TMR1 module of the
                     0006 ; PIC16Cxx family. A LCD display module is used to display (update) the
                     0007 ; time every second. Three keys are used to set the time.
                     0008 ;
                     0009 ;       Program = CLOCK.ASM
                     0010 ;       Revision Date:   5-15-94
                     0011 ;
                     0012 ;*****************************************************************************
                     0013 ;
                     0014 ;
                     0015 ; HARDWARE SETUP
                     0016 ;   LCD Control Lines
                     0017 ;       RA0 = E     (Enable)
                     0018 ;       RA1 = R_W   (Read/Write)
                     0019 ;       RA2 = RS    (Register Select)
                     0020 ;   LCD Data Lines
                     0021 ;       RB<3:0>
                     0022 ;   Switch Inputs
                     0023 ;       RB7 = Select Hour / Minute / Off
                     0024 ;       RB6 = Increment Hour / Minute
                     0025 ;       RB5 = Reset Minutes to 00
                     0026 ;
                     0027             INCLUDE <C74_reg.h>
                     0233
                     0027
                     0028             INCLUDE <CLOCK.h>
                     0028
                     0029 ;
0006                 0030 LCD_DATA        EQU     PORTB       ; The LCD data is on the lower 4-bits
0086                 0031 LCD_DATA_TRIS   EQU     TRISB       ; The TRIS register for the LCD data
0005                 0032 LCD_CNTL        EQU     PORTA       ; Three control lines
                     0033 ;
0000                 0034 PICMaster       EQU     FALSE        ; A Debugging Flag
0000                 0035 Debug           EQU     FALSE        ; A Debugging Flag
```


© 1994 Microchip Technology Inc.

```
0001                    0036 Debug_PU        EQU     TRUE          ; A Debugging Flag
                        0037 ;
                        0038 ;
                        0039 ; Reset address. Determine type of RESET
                        0040 ;
                        0041         org     RESET_V               ; RESET vector location
0000 1683               0042 RESET       BSF     STATUS, RP0       ; Bank 1
0001 188E               0043             BTFSC   PCON, POR         ; Power-up reset?
0002 290C               0044             GOTO    START             ; YES
0003 295F               0045             GOTO    OTHER_RESET       ; NO, a WDT or MCLR reset
                        0046 ;
                        0047 ; This is the Periperal Interrupt routine. Need to determine the type
                        0048 ; of interrupt that occurred. The following interrupts are enabled:
                        0049 ;   1.  PORTB Change (RBIF)
                        0050 ;   2.  TMR1 Overflow Interrupt (T1IF)
                        0051 ;
                        0053         org     ISR_V                 ; Interrupt vector location
                        0054 PER_INT_V
                        0055         if ( Debug )
                        0056             bsf     PORTD, 0          ; Set high, use to measure total
                        0057         endif                         ;    time in Int Service Routine
                        0058 ;
0004 1283               0059             BCF     STATUS, RP0       ; Bank 0
0005 180C               0060             BTFSC   PIR1, TMR1IF      ; Timer 1 overflowed?
0006 2843               0061             GOTO    T1_OVRFL          ; YES, Service the Timer1 Overflow Interrupt
0007 1C0B               0062             BTFSS   INTCON, RBIF      ; NO, Did PORTB change?
0008 28D0               0063             GOTO    ERROR1            ; NO, Error Condition - Unknown Interrupt
                        0064 ;
                        0065 PORTB_FLAG                           ; Are any of PORTB's inputs active?
0009 0806               0066             MOVF    PORTB, W          ;
000A 39E0               0067             ANDLW   0xE0              ; Keep only the 3 switch values
000B 00B5               0068 DEBOUNCE    MOVWF   TEMP              ;
000C 3002               0069             MOVLW   DB_HI_BYTE        ; This is the debounce delay
000D 08B3               0070             MOVF    MSD               ;
000E 01B4               0071             CLRF    LSD               ;
000F 0BB4               0072 KB_D_LP1    DECFSZ  LSD               ;
0010 280F               0073             GOTO    KB_D_LP1          ;
0011 0BB3               0074             DECFSZ  MSD               ;
0012 280F               0075             GOTO    KB_D_LP1          ;
0013 0806               0076 END_DELAY   MOVF    PORTB, W          ;
0014 39E0               0077             ANDLW   0xE0              ; Keep only the 3 switch values
0015 02B5               0078             SUBWF   TEMP, F           ;
0016 1D03               0079             BTFSS   STATUS, Z         ; Is the Zero bit set?
                        0080             ;                         ;   (switches were the same on 2 reads)
0017 280B               0081             GOTO    DEBOUNCE          ; NO, Try another read
0018 00B5               0082 KEY_MATCH   MOVWF   TEMP              ; YES, need to see which is depressed.
                        0083 ;
0019 3080               0084             MOVLW   0x80              ; Since doing key inputs, clear TMR1
```

© 1994 Microchip Technology Inc.

```
001A 008F  0085             MOVWF   TMR1H           ;   for 1 sec overflow.
001B 018E  0086             CLRF    TMR1L           ;
001C 100C  0087             BCF     PIR1, TMR1IF    ; Clear Timer 1 Interrupt Flag
           0088
001D 1FB5  0089             BTFSS   TEMP, HR_MIN_SW ; Is the hour-min-off switch depressed?
001E 2826  0090             GOTO    SELECT_UNITS    ; YES, specify the units selected
001F 1F35  0091             BTFSS   TEMP, INC_SW    ; Is the inc switch depressed?
0020 282B  0092             GOTO    INC_UNIT        ; YES, Increment the selected Units
0021 1EB5  0093             BTFSS   TEMP, CLR_MIN_SW ; Is the clear minute switch depressed?
0022 2835  0094             GOTO    CLR_MIN         ; YES, clear the minutes.
           0095 ;
           0096 ; No key match occured, or finished with PortB interrupt and need to clear interrupt condition.
           0097 ;
           0098 CLR_RB                         ; No RB<7:5> keys are depressed (rising edge Int.)
0023 0886  0099             MOVF    PORTB, F        ; Clear the PORTB mismatch condition
0024 100B  0100             BCF     INTCON, RBIF    ; Clear the PORTB Int Flag
           0101         if ( Debug )
           0102             bcf     PORTD, 0    ; Set low, use to measure total
           0103         endif                   ;   time in Int Service Routine
0025 0009  0104             RETFIE              ; Return / Enable Global Interrupts
           0105 ;
           0107 SELECT_UNITS
0026 30FF  0108             MOVLW   0xFF            ;
0027 00C0  0109             MOVWF   WAIT_CNTR       ; WAIT_CNTR has LSb set after each SELECT UNIT key press.
0028 0AA0  0110             INCF    FLAG_REG, F     ; Increment the pointer to the MIN_UNIT:HR_UNIT
0029 1620  0111             BSF     FLAG_REG, KEY_INPUT ;
002A 2875  0112             GOTO    DISPLAY         ; Flash the Display of the selected unit
           0113 ;
           0114 INC_UNIT
002B 01C0  0115             CLRF    WAIT_CNTR       ; WAIT_CNTR is cleared to zero after each key press.
002C 1820  0116             BTFSC   FLAG_REG, HR_UNIT   ; Are the hour units selected?
002D 285C  0117             GOTO    INC_HRS         ; YES, Increment the hour units
002E 1CA0  0118             BTFSS   FLAG_REG, MIN_UNIT  ; Are the minute units selected?
002F 2823  0119             GOTO    CLR_RB          ; NO, Not a valid key. Clear flags
           0120 ;
0030 0AB1  0121             INCF    MIN, F          ; YES, Increment the minute units
0031 303C  0122             MOVLW   0x3C            ; This is Decimal 60
0032 0231  0123             SUBWF   MIN, W          ; MIN - 60 = ?
0033 1D03  0124             BTFSS   STATUS, Z       ; MIN = 60?
0034 2875  0125             GOTO    DISPLAY         ; NO, display time
           0126                                     ; YES, MIN = 0 (use code from CLR_MIN)
0035 01B1  0127 CLR_MIN     CLRF    MIN             ; MIN = 0
0036 3004  0128             MOVLW   0x04            ; Clear the seconds
0037 00B2  0129             MOVWF   SECS            ; Initial Second count = 4
0038 3080  0130             MOVLW   0x80            ; Clear Timer 1, for 1 sec overflow
0039 008F  0131             MOVWF   TMR1H           ;
003A 018E  0132             CLRF    TMR1L           ;
003B 100C  0133             BCF     PIR1, TMR1IF    ; Clear the TMR1 overflow interrupt.
```


© 1994 Microchip Technology Inc.

```
003C 01C0          0134              CLRF    WAIT_CNTR           ; WAIT_CNTR is cleared to zero after each key press.
003D 1AB5          0135              BTFSC   TEMP, CLR_MIN_SW    ; Is the clear minute switch depressed?
003E 2875          0136              GOTO    DISPLAY             ; NO. Rollover from increment key
003F 10A0          0137              BCF     FLAG_REG, MIN_UNIT  ; YES, Clear ALL relevant flags
0040 1020          0138              BCF     FLAG_REG, HR_UNIT   ;
0041 1220          0139              BCF     FLAG_REG, KEY_INPUT ;
0042 2875          0140              GOTO    DISPLAY             ;
                   0141 ;
                   0143 ;
                   0144 T1_OVRFL
0043 100C          0145              BCF     PIR1, TMR1IF          ; Clear Timer 1 Interrupt Flag
0044 1E20          0146              BTFSS   FLAG_REG, KEY_INPUT   ; Are we using the key inputs?
0045 284F          0147              GOTO    INC_TIME              ; NO, Need to Increment the time
0046 0AC0          0148              INCF    WAIT_CNTR, F          ; YES,
0047 300A          0149              MOVLW   0x0A                  ; 10 counts x 1 seconds
0048 0240          0150              SUBWF   WAIT_CNTR, W          ; Has the 10 Sec wait for key expired?
0049 1D03          0151              BTFSS   STATUS, Z             ; Is the result 0?
004A 2875          0152              GOTO    DISPLAY               ; NO, Display value
004B 01C0          0153              CLRF    WAIT_CNTR             ; YES, Clear WAIT_CNTR
004C 1220          0154              BCF     FLAG_REG, KEY_INPUT   ;
004D 1020          0155              BCF     FLAG_REG, HR_UNIT     ;
004E 10A0          0156              BCF     FLAG_REG, MIN_UNIT    ;
                   0157 ;
                   0158 ;
004F 3080          0159 INC_TIME     MOVLW   0x80            ;
0050 008F          0160              MOVWF   TMR1H           ; 1 Second Overflow
0051 0AB2          0161              INCF    SECS, F         ;
0052 1F32          0162              BTFSS   SECS, 6         ;
0053 2875          0163              GOTO    DISPLAY         ;
0054 3004          0164              MOVLW   0x04            ;
0055 00B2          0165              MOVWF   SECS            ;
0056 0AB1          0166              INCF    MIN, F          ;
0057 303C          0167              MOVLW   0x3C            ; W = 60d
0058 0231          0168              SUBWF   MIN, W          ;
0059 1D03          0169              BTFSS   STATUS, Z       ;
005A 2875          0170              GOTO    DISPLAY         ;
005B 01B1          0171              CLRF    MIN             ;
005C 0AB0          0172 INC_HRS      INCF    HRS, F          ;
005D 300C          0173              MOVLW   0x0C            ; It is now 12:00, Toggle AM / PM
005E 0230          0174              SUBWF   HRS, W          ;
005F 1D03          0175              BTFSS   STATUS, Z       ;
0060 2867          0176              GOTO    CK_13           ; Need to check if HRS = 13
0061 1FA0          0177              BTFSS   FLAG_REG, AM    ; Was it AM or PM
0062 2865          0178              GOTO    SET_AM          ; Was PM, Needs to be AM
0063 13A0          0179              BCF     FLAG_REG, AM    ; It is PM
0064 2875          0180              GOTO    DISPLAY         ;
0065 17A0          0181 SET_AM       BSF     FLAG_REG, AM    ; It is AM
0066 2875          0182              GOTO    DISPLAY         ;
```

© 1994 Microchip Technology Inc.

```
                        0183 ;
0067 300D               0184 CK_13       MOVLW   0x0D            ; Check if HRS = 13
0068 0230               0185             SUBWF   HRS, W          ;
0069 1D03               0186             BTFSS   STATUS, Z       ;
006A 2875               0187             GOTO    DISPLAY         ;
006B 01B0               0188             CLRF    HRS             ;
006C 0AB0               0189             INCF    HRS, F          ;
006D 2875               0190             GOTO    DISPLAY         ;
                        0191 ;
                        0193 INIT_DISPLAY
006E 300C               0194             MOVLW   DISP_ON         ; Display On, Cursor On
006F 20E3               0195             CALL    SEND_CMD        ; Send This command to the Display Module
0070 3001               0196             MOVLW   CLR_DISP        ; Clear the Display
0071 20E3               0197             CALL    SEND_CMD        ; Send This command to the Display Module
0072 3006               0198             MOVLW   ENTRY_INC       ; Set Entry Mode Inc., No shift
0073 20E3               0199             CALL    SEND_CMD        ; Send This command to the Display Module
0074 0008               0200             RETURN
                        0201 ;
                        0202 DISPLAY
0075 3080               0203             MOVLW   DD_RAM_ADDR     ;
0076 20E3               0204             CALL    SEND_CMD        ;
                        0205 ;
0077 1A20               0206             BTFSC   FLAG_REG, KEY_INPUT ; Do we need to flash the selectected units?
0078 287D               0207             GOTO    FLASH_UNITS         ; YES, we need to flash selected units
0079 20A4               0208             CALL    LOAD_HRS            ; NO, do a normal display
007A 20AD               0209             CALL    LOAD_COLON          ;
007B 20B2               0210             CALL    LOAD_MIN            ;
007C 28BB               0211             GOTO    LOAD_AM             ;
                        0212 ;
                        0213 FLASH_UNITS
007D 018A               0214             CLRF    PCLATH              ; This clears PCLATH, This table in 1st
007E 0820               0215             MOVF    FLAG_REG, W         ;    256 bytes of program memory
007F 3903               0216             ANDLW   0x03                ; only HR_UNIT and MIN_UNIT bit can be non-zero
                        0217 UNIT_TBL
0080 0782               0218             ADDWF   PCL                 ; HR_UNIT:MIN_UNIT
0081 289F               0219             GOTO    NO_UNITS            ;    0 0  - Display everything.
0082 2887               0220             GOTO    HR_UNITS            ;    0 1  - Flash the hour units
0083 2893               0221             GOTO    MIN_UNITS           ;    1 0  - Flash the minute units
                        0222 UNIT_TBL_END
0084 30FC               0223             MOVLW   0xFC                ;    1 1  - Need to clear FLAG_REG
0085 05A0               0224             ANDWF   FLAG_REG, F         ;                    <HR_UNIT:MIN_UNIT>
0086 289F               0225             GOTO    NO_UNITS            ;    0 0  - Display everything.
                        0226 ;
                        0227     if ( (UNIT_TBL & 0x0FF) >= (UNIT_TBL_END & 0x0FF) )
                        0228        MESSG   "Warning: Table UNIT_TBL crosses page boundry in computed jump"
                        0229     endif
                        0230 ;
                        0231 ;
```


© 1994 Microchip Technology Inc.

```
                        0232 HR_UNITS
0087 1C40               0233            BTFSS   WAIT_CNTR, 0    ; If WAIT_CNTR is odd,
                        0234                                    ;    hour digits are displayed as blank
0088 288D               0235            GOTO    SKIP_BLK_HRS    ;
0089 3020               0236            MOVLW   ' '             ;
008A 20D4               0237            CALL    SEND_CHAR       ;
008B 3020               0238            MOVLW   ' '             ;
008C 20D4               0239            CALL    SEND_CHAR       ;
                        0240 SKIP_BLK_HRS
008D 1C40               0241            BTFSS   WAIT_CNTR, 0    ; WAIT_CNTR was even, display hour digits
008E 20A4               0242            CALL    LOAD_HRS        ;
                        0243 ;
008F 303A               0244            MOVLW   ':'             ; : always on, display all other character
0090 20D4               0245            CALL    SEND_CHAR       ;
0091 20B2               0246            CALL    LOAD_MIN        ;
0092 28BB               0247            GOTO    LOAD_AM         ;
                        0248 ;
                        0250 MIN_UNITS
0093 20A4               0251            CALL    LOAD_HRS        ; Display hours
0094 303A               0252            MOVLW   ':'             ; : always on
0095 20D4               0253            CALL    SEND_CHAR       ;
0096 1C40               0254            BTFSS   WAIT_CNTR, 0    ; If WAIT_CNTR is odd,
                        0255                                    ;   minute digits are displayed as blank
0097 289C               0256            GOTO    SKIP_BLK_MIN    ;
0098 3020               0257            MOVLW   ' '             ;
0099 20D4               0258            CALL    SEND_CHAR       ;
009A 3020               0259            MOVLW   ' '             ;
009B 20D4               0260            CALL    SEND_CHAR       ;
                        0261 SKIP_BLK_MIN
009C 1C40               0262            BTFSS   WAIT_CNTR, 0    ; WAIT_CNTR was even, display minute digits
009D 20B2               0263            CALL    LOAD_MIN        ;
009E 28BB               0264            GOTO    LOAD_AM         ;
                        0265 ;
                        0266 NO_UNITS
009F 20A4               0267            CALL    LOAD_HRS        ; Display all character
00A0 303A               0268            MOVLW   ':'             ;
00A1 20D4               0269            CALL    SEND_CHAR       ;
00A2 20B2               0270            CALL    LOAD_MIN        ;
00A3 28BB               0271            GOTO    LOAD_AM         ;
                        0272 ;
                        0273 LOAD_HRS
00A4 0830               0274            MOVF    HRS, W          ; Load the Wreg with the value
00A5 20C7               0275            CALL    BIN_2_BCD       ;   to convert to BCD
00A6 0833               0276            MOVF    MSD, W          ; Load the MSD value into the Wreg
00A7 2400               0277            CALL    NUM_TABLE       ; Get the ASCII code
00A8 20D4               0278            CALL    SEND_CHAR       ; Send this Character to the Display
                        0279 ;
00A9 0834               0280            MOVF    LSD, W          ; Load the LSD value into the Wreg
```


© 1994 Microchip Technology Inc.

```
00AA 2400  0281              CALL    NUM_TABLE       ; Get the ASCII code
00AB 20D4  0282              CALL    SEND_CHAR       ; Send this Character to the Display
00AC 0008  0283              RETURN
           0284 ;
00AD 3020  0285 LOAD_COLON   MOVLW   ' '             ; ASCII value for a Blank space
00AE 1832  0286              BTFSC   SECS, 0         ; Is it an EVEN or ODD second
00AF 3E1A  0287              ADDLW   ':' - ' '       ; Is ODD, Second colon is ON.
           0288                                      ;   Add delta offset of ASCII Characters
00B0 20D4  0289              CALL    SEND_CHAR       ; Send this Character to the Display
00B1 0008  0290              RETURN
           0291 ;
           0292 LOAD_MIN
00B2 0831  0293              MOVF    MIN, W          ; Load the Wreg with the value
00B3 20C7  0294              CALL    BIN_2_BCD       ;   to convert to BCD
00B4 0833  0295              MOVF    MSD, W          ; Load the MSD value into the Wreg
00B5 2400  0296              CALL    NUM_TABLE       ; Get the ASCII code
00B6 20D4  0297              CALL    SEND_CHAR       ; Send this Character to the Display
           0298 ;
00B7 0834  0299              MOVF    LSD, W          ; Load the LSD value into the Wreg
00B8 2400  0300              CALL    NUM_TABLE       ; Get the ASCII code
00B9 20D4  0301              CALL    SEND_CHAR       ; Send this Character to the Display
00BA 0008  0302              RETURN
           0303 ;
00BB 3020  0305 LOAD_AM      MOVLW   ' '             ; ASCII value for a Blank space
00BC 20D4  0306              CALL    SEND_CHAR       ; Send this Character to the Display
00BD 3041  0307              MOVLW   'A'             ; ASCII value for a Blank space
00BE 1FA0  0308              BTFSS   FLAG_REG, AM    ; Is it AM or PM
00BF 3E0F  0309              ADDLW   'P' - 'A'       ; Is PM, Add delta offset of ASCII Characters
00C0 20D4  0310              CALL    SEND_CHAR       ; Send this Character to the Display
00C1 304D  0311              MOVLW   'M'
00C2 20D4  0312              CALL    SEND_CHAR       ; Send this Character to the Display
           0313 ;
00C3 1683  0314              BSF     STATUS, RP0     ; Bank 1
00C4 1381  0315              BCF     OPTION_R, RBPU  ; Turn on PORTB Pull-up
00C5 1283  0316              BCF     STATUS, RP0     ; Bank 0
00C6 2823  0317              GOTO    CLR_RB          ; You've displayed the time, Clear RBIF
           0318 ;
           0319 ;
           0320 ;*************************************************************************
           0321 ; The BIN_2_BCD routine converts the binary number, in the W register, to a
           0322 ; binary coded decimal (BCD) munber. This BCD number is stored MSD:LSD. This
           0323 ; routine is used by the DISPLAY subroutine, to convert the time values.
           0324 ;*************************************************************************
           0325 ;
00C7 01B3  0326 BIN_2_BCD    CLRF    MSD             ; This value contain the 10's digit value
00C8 00B4  0327              MOVWF   LSD             ; This value contain the 1's digit value
00C9 300A  0328 TENS_SUB     MOVLW   .10             ; A decimal 10
00CA 0234  0329              SUBWF   LSD, W          ;
00CB 1C03  0330              BTFSS   STATUS, C       ; Did this subtract cause a Negative Result?
```


© 1994 Microchip Technology Inc.

```
00CC 3400          0331            RETLW   0               ; YES, Return from this Routine
00CD 00B4          0332            MOVWF   LSD             ; No, move the result into LSD
00CE 0AB3          0333            INCF    MSD, F          ; Increment the most significat digit
00CF 28C9          0334            GOTO    TENS_SUB        ;
                   0335 ;
                   0336 ;
                   0337 ; Should NEVER get here
                   0338 ;
00D0 1283          0339 ERROR1     BCF     STATUS, RP0         ; Bank 0
                   0340 ;
                   0341        if ( Debug )
                   0342            bsf     PORTD, 1
                   0343            bcf     PORTD, 1
                   0344        else
00D1 1407          0345            BSF     PORTC, 0
00D2 1007          0346            BCF     PORTC, 0
                   0347        endif
00D3 28D0          0348            GOTO    ERROR1
                   0349 ;
                   0351 ;
                   0352 ;*****************************************************************
                   0353 ;* SendChar - Sends character to LCD                             *
                   0354 ;* This routine splits the character into the upper and lower    *
                   0355 ;* nibbles and sends them to the LCD, upper nibble first.        *
                   0356 ;* The data is transmitted on the PORT<3:0> pins                 *
                   0357 ;*****************************************************************
                   0358
                   0359 SEND_CHAR
00D4 00B6          0360            MOVWF   CHAR            ; Character to be sent is in W
00D5 20F2          0361            CALL    BUSY_CHECK      ; Wait for LCD to be ready
00D6 0E36          0362            SWAPF   CHAR, W
00D7 390F          0363            ANDLW   0x0F            ; Get upper nibble
00D8 0086          0364            MOVWF   LCD_DATA        ; Send data to LCD
00D9 1085          0365            BCF     LCD_CNTL, R_W   ; Set LCD to read
00DA 1505          0366            BSF     LCD_CNTL, RS    ; Set LCD to data mode
00DB 1405          0367            BSF     LCD_CNTL, E     ; toggle E for LCD
00DC 1005          0368            BCF     LCD_CNTL, E
00DD 0836          0369            MOVF    CHAR, W
00DE 390F          0370            ANDLW   0x0F            ; Get lower nibble
00DF 0086          0371            MOVWF   LCD_DATA        ; Send data to LCD
00E0 1405          0372            BSF     LCD_CNTL, E     ; toggle E for LCD
00E1 1005          0373            BCF     LCD_CNTL, E
00E2 0008          0374            RETURN
                   0375
                   0376 ;*****************************************************************
```

© 1994 Microchip Technology Inc.

```
                        0377 ;* SendCmd - Sends command to LCD                                *
                        0378 ;* This routine splits the command into the upper and lower      *
                        0379 ;* nibbles and sends them to the LCD, upper nibble first.        *
                        0380 ;* The data is transmitted on the PORT<3:0> pins                 *
                        0381 ;*************************************************************
                        0382
                        0383 SEND_CMD
00E3 00B6               0384             MOVWF   CHAR            ; Character to be sent is in W
00E4 20F2               0385             CALL    BUSY_CHECK      ; Wait for LCD to be ready
00E5 0E36               0386             SWAPF   CHAR, W
00E6 390F               0387             ANDLW   0x0F            ; Get upper nibble
00E7 0086               0388             MOVWF   LCD_DATA        ; Send data to LCD
00E8 1085               0389             BCF     LCD_CNTL, R_W   ; Set LCD to read
00E9 1105               0390             BCF     LCD_CNTL, RS    ; Set LCD to command mode
00EA 1405               0391             BSF     LCD_CNTL, E     ; toggle E for LCD
00EB 1005               0392             BCF     LCD_CNTL, E
00EC 0836               0393             MOVF    CHAR, W
00ED 390F               0394             ANDLW   0x0F            ; Get lower nibble
00EE 0086               0395             MOVWF   LCD_DATA        ; Send data to LCD
00EF 1405               0396             BSF     LCD_CNTL, E     ; toggle E for LCD
00F0 1005               0397             BCF     LCD_CNTL, E
00F1 0008               0398             RETURN
                        0400 ;*************************************************************
                        0401 ;* This routine checks the busy flag, returns when not busy     *
                        0402 ;*  Affects:                                                    *
                        0403 ;*      TEMP - Returned with busy/address                       *
                        0404 ;*************************************************************
                        0405
                        0406 BUSY_CHECK
                        0407 ;
                        0408         if ( Debug )
                        0409             bsf     PORTD, 3
                        0410             bcf     PORTD, 3
                        0411         endif
00F2 0186               0412             CLRF    LCD_DATA        ;** Have PORTB<3:0> output low
00F3 1683               0413             BSF     STATUS, RP0     ; Bank 1
00F4 1781               0414             BSF     OPTION_R, RBPU  ; Turn off PORTB Pull-up
00F5 30FF               0415             MOVLW   0xFF            ; Set PortB for input
00F6 0086               0416             MOVWF   LCD_DATA_TRIS
00F7 1283               0417             BCF     STATUS, RP0     ; Bank 0
00F8 1105               0418             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
00F9 1485               0419             BSF     LCD_CNTL, R_W   ; Setup to read busy flag
00FA 1405               0420             BSF     LCD_CNTL, E     ; Set E high
00FB 1005               0421             BCF     LCD_CNTL, E     ; Set E low
00FC 0E06               0422             SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
```


© 1994 Microchip Technology Inc.

```
00FD 39F0  0423            ANDLW   0xF0            ; Mask out lower nibble
00FE 00B5  0424            MOVWF   TEMP            ;
00FF 1405  0425            BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0100 1005  0426            BCF     LCD_CNTL, E
0101 0806  0427            MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
0102 390F  0428            ANDLW   0x0F            ; Mask out upper nibble
0103 04B5  0429            IORWF   TEMP, F         ; Combine nibbles
0104 1BB5  0430            BTFSC   TEMP, 7         ; Check busy flag, high = busy
0105 28F2  0431            GOTO    BUSY_CHECK      ; If busy, check again
0106 1085  0432            BCF     LCD_CNTL, R_W
0107 1683  0433            BSF     STATUS, RP0     ; Bank 1
0108 30F0  0434            MOVLW   0xF0            ;
0109 0086  0435            MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
010A 1283  0436            BCF     STATUS, RP0     ; Bank 0
010B 0008  0437            RETURN
           0438 ;
           0440 ;
           0441 ;*******************************************************************************
           0442 ;*****      Start program here, Power-On Reset occurred.
           0443 ;*******************************************************************************
           0444 ;
           0445 START                             ; POWER_ON Reset (Beginning of program)
010C 1283  0446            BCF     STATUS, RP0     ; Bank 0
010D 300C  0447            MOVLW   0x0C            ; Decimal 12
010E 00B0  0448            MOVWF   HRS             ; HOURS = 12
010F 01B1  0449            CLRF    MIN             ; MIN   = 00
0110 3000  0450            MOVLW   0x00            ;
0111 00A0  0451            MOVWF   FLAG_REG        ; PM light is on
0112 3004  0452            MOVLW   0x04            ; Initial value of seconds (64d - 60d)
0113 00B2  0453            MOVWF   SECS            ; This allows a simple bit test to see if 60
           0454                                    ;   secs has elapsed.
0114 3080  0455            MOVLW   0x80            ; TIM1H:TMR1L = 0x8000 gives 1 second
0115 008F  0456            MOVWF   TMR1H           ;   overflow, at 32 KHz.
0116 018E  0457            CLRF    TMR1L           ;
           0458 ;
           0459 MCLR_RESET                        ; A Master Clear Reset
0117 1283  0460            BCF     STATUS, RP0     ; Bank 0
0118 0183  0461            CLRF    STATUS          ; Do initialization (Bank 0)
0119 018B  0462            CLRF    INTCON
011A 018C  0463            CLRF    PIR1
011B 1683  0464            BSF     STATUS, RP0     ; Bank 1
011C 3000  0465            MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
011D 0081  0466            MOVWF   OPTION_R        ;
011E 018C  0467            CLRF    PIE1            ; Disable all peripheral interrupts
011F 30FF  0468            MOVLW   0xFF            ;
0120 009F  0469            MOVWF   ADCON1          ; Port A is Digital.
           0470 ;
           0471 ;
```

© 1994 Microchip Technology Inc.

```
0121 1283          0472              BCF     STATUS, RP0     ; Bank 0
0122 0185          0473              CLRF    PORTA           ; ALL PORT output should output Low.
0123 0186          0474              CLRF    PORTB
0124 0187          0475              CLRF    PORTC
0125 0188          0476              CLRF    PORTD
0126 0189          0477              CLRF    PORTE
0127 1010          0478              BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                   0479 ;
0128 1683          0480              BSF     STATUS, RP0     ; Select Bank 1
0129 0185          0481              CLRF    TRISA           ; RA5 -  0 outputs
012A 30F0          0482              MOVLW   0xF0            ;
012B 0086          0483              MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
012C 0187          0484              CLRF    TRISC           ; RC Port are outputs
012D 1407          0485              BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
012E 0188          0486              CLRF    TRISD           ; RD Port are outputs
012F 0189          0487              CLRF    TRISE           ; RE Port are outputs
0130 140C          0488              BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0131 1381          0489              BCF     OPTION_R, RBPU  ; Enable PORTB pull-ups
0132 1283          0490              BCF     STATUS, RP0     ; Select Bank 0
0133 0886          0491              MOVF    PORTB, F        ; Need to clear 1st RBIF, due to
0134 100B          0492              BCF     INTCON, RBIF    ;   set up of PORTB
                   0493 ;
                   0495 ;
                   0496 ; Initilize the LCD Display Module
                   0497 ;
0135 0185          0498              CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                   0499
                   0500 DISPLAY_INIT
0136 3002          0501              MOVLW   0x02            ; Command for 4-bit interface
0137 0086          0502              MOVWF   LCD_DATA        ;
0138 1405          0503              BSF     LCD_CNTL, E     ;
0139 1005          0504              BCF     LCD_CNTL, E     ;
                   0505 ;
                   0506 ; This routine takes the calculated times that the delay loop needs to
                   0507 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                   0508 ; frequency of operation. These uses registers before they are needed to
                   0509 ; store the time.
                   0510 ;
013A 3006          0511 LCD_DELAY    MOVLW   LCD_INIT_DELAY  ;
013B 00B3          0512              MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
013C 01B4          0513              CLRF    LSD             ;
013D 0BB4          0514 LOOP2        DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
013E 293D          0515              GOTO    LOOP2           ;
013F 0BB3          0516              DECFSZ  MSD             ;
                   0517 END_LCD_DELAY
0140 293D          0518              GOTO    LOOP2           ;
                   0519 ;
                   0520 ; Command sequence for 2 lines of 5x7 characters
```


© 1994 Microchip Technology Inc.

```
                        0521 ;
0141 3002               0522 CMD_SEQ    MOVLW   0X02
0142 0086               0523            MOVWF   LCD_DATA
0143 1405               0524            BSF     LCD_CNTL, E     ;
0144 1005               0525            BCF     LCD_CNTL, E     ;
0145 3008               0526            MOVLW   0x08            ;
0146 0086               0527            MOVWF   LCD_DATA        ;
0147 1405               0528            BSF     LCD_CNTL, E     ;
0148 1005               0529            BCF     LCD_CNTL, E     ;
                        0530 ;
                        0531 ; Busy Flag should be valid after this point
                        0532 ;
0149 300C               0533            MOVLW   DISP_ON         ;
014A 20E3               0534            CALL    SEND_CMD        ;
014B 3001               0535            MOVLW   CLR_DISP        ;
014C 20E3               0536            CALL    SEND_CMD        ;
014D 3006               0537            MOVLW   ENTRY_INC       ;
014E 20E3               0538            CALL    SEND_CMD        ;
014F 3080               0539            MOVLW   DD_RAM_ADDR     ;
0150 20E3               0540            CALL    SEND_CMD        ;
                        0541 ;
                        0543 ;
                        0544 ; Initialize the Special Function Registers (SFR) interrupts
                        0545 ;
0151 018C               0546            CLRF    PIR1            ;
0152 300E               0547            MOVLW   0x0E
0153 0090               0548            MOVWF   T1CON           ; RC1 is overridden by TCKO
0154 170B               0549            BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0155 158B               0550            BSF     INTCON, RBIE    ; Disable PORTB<7:4> Change Interrupts
0156 178B               0551            BSF     INTCON, GIE     ; Enable all Interrupts
                        0552 ;
0157 206E               0553            CALL    INIT_DISPLAY    ;
0158 2075               0554            CALL    DISPLAY         ;
                        0555 ;
0159 300E               0556            MOVLW   0x0E
015A 0090               0557            MOVWF   T1CON           ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
015B 1410               0558            BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
                        0559 ;
                        0560        if ( PICMaster )
                        0561 lzz        goto    lzz             ; Loop waiting for interrupts (for use with PICMASTER)
                        0562        else
                        0563 ;
015C 0063               0564 SLEEP_LP   SLEEP                   ; Wait for Change on PORTB interrupt. or TMR1 timeout
015D 0000               0565            NOP                     ;
015E 295C               0566            GOTO    SLEEP_LP        ;
                        0567 ;
                        0568        endif
                        0569 ;
```

© 1994 Microchip Technology Inc.

```
                    0570 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                    0571 ;
015F 1E03           0572 OTHER_RESET   BTFSS  STATUS, TO     ; WDT Time-out?
0160 28D0           0573 WDT_TIMEOUT   GOTO   ERROR1         ; YES, This is error condition
                    0574         if ( Debug_PU )
0161 290C           0575             goto   START           ; MCLR reset, Goto START
                    0576         else
                    0577             GOTO   MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                    0578         endif
                    0579 ;
                    0580         if (Debug )
                    0581 END_START     NOP                   ; END lable for debug
                    0582         endif
                    0583 ;
                    0585 ;
                    0586     org     TABLE_ADDR
                    0587 ;
0400 00B5           0588 NUM_TABLE      MOVWF   TEMP              ; Store value to TEMP register
0401 3004           0589                MOVLW   HIGH (TABLE_ADDR) ; Ensure that the PCLATH high has the
0402 008A           0590                MOVWF   PCLATH            ;  correct value
0403 0835           0591                MOVF    TEMP, W           ; Value into table
0404 390F           0592                ANDLW   0x0F              ; Mask to 4-bits (00 - 0Fh)
0405 0782           0593 NUM_TBL        ADDWF   PCL, F            ; Determine Offset into table
0406 3430           0594                RETLW   '0'               ; ASCII value of "0" in W register
0407 3431           0595                RETLW   '1'               ; ASCII value of "1" in W register
0408 3432           0596                RETLW   '2'               ; ASCII value of "2" in W register
0409 3433           0597                RETLW   '3'               ; ASCII value of "3" in W register
040A 3434           0598                RETLW   '4'               ; ASCII value of "4" in W register
040B 3435           0599                RETLW   '5'               ; ASCII value of "5" in W register
040C 3436           0600                RETLW   '6'               ; ASCII value of "6" in W register
040D 3437           0601                RETLW   '7'               ; ASCII value of "7" in W register
040E 3438           0602                RETLW   '8'               ; ASCII value of "8" in W register
040F 3439           0603                RETLW   '9'               ; ASCII value of "9" in W register
                    0604                                          ; Any enter after is in error (Display an E)
0410 3445           0605                RETLW   'E'               ; ASCII value of "E" in W register
0411 3445           0606                RETLW   'E'               ; ASCII value of "E" in W register
0412 3445           0607                RETLW   'E'               ; ASCII value of "E" in W register
0413 3445           0608                RETLW   'E'               ; ASCII value of "E" in W register
0414 3445           0609                RETLW   'E'               ; ASCII value of "E" in W register
0415 3445           0610 NUM_TBL_END    RETLW   'E'               ; ASCII value of "E" in W register
                    0611 ;
                    0612     if ( (NUM_TBL & 0x0FF) >= (NUM_TBL_END & 0x0FF) )
                    0613        MESSG   "Warning: Table NUM_TBL crosses page boundry in computed jump"
                    0614     endif
                    0615 ;
                    0616 ;
                    0617     org     PMEM_END      ; End of Program Memory
07FF 28D0           0618     GOTO    ERROR1          ; If you get here your program was lost
```


© 1994 Microchip Technology Inc.

```
                    0619
                    0620     end
                    0621
                    0622


MPASM 01.01 Released     CLOCK.ASM   5-13-1994  13:11:9                PAGE 15


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XX-------- --------

0400 : XXXXXXXXXXXXXXXX XXXXXX---------- -------- --------
0440 : -------- -------- -------- --------

0780 : -------- -------- -------- --------
07C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :     0
Warnings :    16
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

© 1994 Microchip Technology Inc.

APPENDIX B: CLOCK_01.H INCLUDE FILE

```
        nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:        CLOCK_01.H
;       Revision:       5-04-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq                EQU     D'4000000'      ; Device Frequency is 4 MHz
DB_HI_BYTE              EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY          EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR              EQU     40              ; RAM Location
OUTER_CNTR              EQU     41              ; RAM Location
;
T1OSO                   EQU     0               ; The RC0 / T1OSO / T1CKI
;
RESET_V                 EQU     0x0000          ; Address of RESET Vector
ISR_V                   EQU     0x0004          ; Address of Interrupt Vector
PMEM_END                EQU     0x07FF          ; Last address in Program Memory
TABLE_ADDR              EQU     0x0400          ; Address where to start Tables
;
HR_MIN_SW               EQU     0x7             ; The switch to select the units
INC_SW                  EQU     0x6             ; The switch to increment the selected units
CLR_MIN_SW              EQU     0x5             ; The switch to clear the minutes and seconds
;
FLAG_REG                EQU     0x020           ; Register which contains flag bits
;
;  +------+---+---+-----------+---+---+----------+---------+
;  |  AM  | - | - | KEY_INPUT | - | - | MIN_UNIT | HR_UNIT |
;  +------+---+---+-----------+---+---+----------+---------+
;
AM                      EQU     0x07            ; Flag to specify if AM or PM
;
KEY_INPUT               EQU     0x04            ; Flag to specify if doing key inputs
;
MIN_UNIT                EQU     0x01            ; Flags to specify which units to operate on
HR_UNIT                 EQU     0x00            ;  (HRS, MIN, or none)
;
HRS                     EQU     0x030           ; Holds counter value for HOURS
MIN                     EQU     0x031           ; Holds counter value for MINUTES
SECS                    EQU     0x032           ; Holds counter value for SECONDS
MSD                     EQU     0x033           ; Temp. register, Holds Most Significant
                                                ;   Digit of BIN to BCD conversion
LSD                     EQU     0x034           ; Temporary register, Holds Least Significant
                                                ;   Digit of BIN to BCD conversion
TEMP                    EQU     0x035           ; Temporary register
CHAR                    EQU     0x036           ; Temporary register,
                                                ;  Holds value to send to LCD module.
;
WAIT_CNTR               EQU     0x040           ; Counter that holds wait time for key inputs
;
; LCD Display Commands and Control Signal names.
;
E                       EQU     0               ; LCD Enable control line
R_W                     EQU     1               ; LCD Read/Write control line
RS                      EQU     2               ; LCD Register Select control line
;
; LCD Module commands
;
DISP_ON                 EQU     0x00C           ; Display on
DISP_ON_C               EQU     0x00E           ; Display on, Cursor on
DISP_ON_B               EQU     0x00F           ; Display on, Cursor on, Blink cursor
```


© 1994 Microchip Technology Inc.

```
DISP_OFF            EQU     0x008       ; Display off
CLR_DISP            EQU     0x001       ; Clear the Display
ENTRY_INC           EQU     0x006       ;
ENTRY_INC_S         EQU     0x007       ;
ENTRY_DEC           EQU     0x004       ;
ENTRY_DEC_S         EQU     0x005       ;
DD_RAM_ADDR         EQU     0x080       ; Least Significant 7-bit are for address
DD_RAM_UL           EQU     0x080       ; Upper Left coner of the Display
;
        list
```

APPENDIX C: C74_REG.H INCLUDE FILE

```
    nolist
;
;   File =   C64_reg.h
;   Rev. History:     08-04-93 by MP
;                     10-18-93 by MP to make Page ok
;                     11-15-93 by MP to have correct pages for SFR
;
; EQUates for Special Function Registers
;
;
INDF                  EQU           00
RTCC                  EQU           01
OPTION_R              EQU           81
PCL                   EQU           02
STATUS                EQU           03
FSR                   EQU           04
PORTA                 EQU           05
TRISA                 EQU           85
PORTB                 EQU           06
TRISB                 EQU           86
PORTC                 EQU           07
TRISC                 EQU           87
PORTD                 EQU           08
TRISD                 EQU           88
PORTE                 EQU           09
TRISE                 EQU           89
PCLATH                EQU           0A
INTCON                EQU           0B
PIR1                  EQU           0C
PIE1                  EQU           8C
TMR1L                 EQU           0E
PCON                  EQU           8E
TMR1H                 EQU           0F
T1CON                 EQU           10
TMR2                  EQU           11
T2CON                 EQU           12
PR2                   EQU           92
SSPBUF                EQU           13
SSPADD                EQU           93
SSPCON                EQU           14
SSPSTAT               EQU           94
CCPR1L                EQU           15
CCPR1H                EQU           16
CCP1CON               EQU           17
RCSTA                 EQU           18
TXSTA                 EQU           98
TXREG                 EQU           19
SPBRG                 EQU           99
RCREG                 EQU           1A
CCPR2L                EQU           1B
CCPR2H                EQU           1C
CCP2CON               EQU           1D
ADRES                 EQU           1E
ADCON0                EQU           1F
ADCON1                EQU           9F

;
;*********************************************
;************   Bit Deffinitions   ************
;*********************************************
;
; STATUS register (Address 03/83)
;
IRP                    EQU           7
RP1                    EQU           6
RP0                    EQU           5
```


© 1994 Microchip Technology Inc.

```
TO                      EQU             4
PD                      EQU             3
Z                       EQU             2
DC                      EQU             1
C                       EQU             0
;
; INTCON register (Address 0B/8B)
;
GIE                     EQU             7
PEIE                    EQU             6
RTIE                    EQU             5
INTE                    EQU             4
RBIE                    EQU             3
RTIF                    EQU             2
INTF                    EQU             1
RBIF                    EQU             0
;
; PIR1 register (Address 0C)
;
PSPIF                   EQU             7
SSPIF                   EQU             3
CCP1IF                  EQU             2
TMR2IF                  EQU             1
TMR1IF                  EQU             0
;
; PIE1 register (Address 8C)
;
PSPIE                   EQU             7
SSPIE                   EQU             3
CCP1IE                  EQU             2
TMR2IE                  EQU             1
TMR1IE                  EQU             0
;
; OPTION register (Address 81)
;
RBPU                    EQU             7
INTEDG                  EQU             6
RTS                     EQU             5
RTE                     EQU             4
PSA                     EQU             3
PS2                     EQU             2
PS1                     EQU             1
PS0                     EQU             0
;
; PCON register (Address 8E)
;
POR                     EQU             1
;
; TRISE register (Address 89)
;
IBF                     EQU             7
OBF                     EQU             6
IBOV                    EQU             5
PSPMODE                 EQU             4
TRISE2                  EQU             2
TRISE1                  EQU             1
TRISE0                  EQU             0

;
; T1CON register (Address 10)
;
T1CKPS1                 EQU             5
T1CKPS0                 EQU             4
T1OSCEN                 EQU             3
T1INSYNC                EQU             2
TMR1CS                  EQU             1
TMR1ON                  EQU             0
;
```


© 1994 Microchip Technology Inc.

```
; T2CON register (Address 12)
;
TOUTPS3                 EQU             6
TOUTPS2                 EQU             5
TOUTPS1                 EQU             4
TOUTPS0                 EQU             3
TMR2ON                  EQU             2
T2CKPS1                 EQU             1
T2CKPS0                 EQU             0
;
; SSPCON register (Address 14)
;
WCOL                    EQU             7
SSPOV                   EQU             6
SSPEN                   EQU             5
CKP                     EQU             4
SSPM3                   EQU             3
SSPM2                   EQU             2
SSPM1                   EQU             1
SSPM0                   EQU             0
;
; SSPSTAT register (Address 94)
;
DA                      EQU             5
P                       EQU             4
S                       EQU             3
RW                      EQU             2
UA                      EQU             1
BF                      EQU             0
;
; CCP1CON register (Address 17)
;
CCP1X                   EQU             5
CCP1Y                   EQU             4
CCP1M3                  EQU             3
CCP1M2                  EQU             2
CCP1M1                  EQU             1
CCP1M0                  EQU             0
;
; RCSTA register (Address 18)
;
SPEN                    EQU             7
RC89                    EQU             6
SREN                    EQU             5
CREN                    EQU             4
FERR                    EQU             2
OERR                    EQU             1
RCD8                    EQU             0
;
; TXSTA register (Address 98)
;
CSRC                    EQU             7
TX89                    EQU             6
TXEN                    EQU             5
SYNC                    EQU             4
BRGH                    EQU             2
TRMT                    EQU             1
TXD8                    EQU             0


;
; CCP2CON register (Address 1D)
;
CCP2X                   EQU             5
CCP2Y                   EQU             4
CCP2M3                  EQU             3
CCP2M2                  EQU             2
CCP2M1                  EQU             1
CCP2M0                  EQU             0
```


© 1994 Microchip Technology Inc.

```
;
; ADCON0 register (Address 1F)
;
ADCS1                   EQU             7
ADCS0                   EQU             6
CHS2                    EQU             5
CHS1                    EQU             4
CHS0                    EQU             3
GO                              EQU             2
DONE                    EQU             2
ADON                    EQU             0
;
; ADCON1 register (Address 9F)
;
PCFG2                   EQU             2
PCFG1                   EQU             1
PCFG0                   EQU             0
;
;*********************************************
;**** Bits for destination control
;****    W = W register is destination
;****    F = File register is destination
;*********************************************
;
W                       EQU             0
F                       EQU             1
;
FALSE                   EQU             0
TRUE                    EQU             1

    list
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN546

Using the Analog to Digital Converter

INTRODUCTION

This application note is intended for PIC16C7X users with various degrees of familiarity with analog system design. The various sections discuss the following topics:

- Commonly used A/D terminology
- How to configure and use the PIC16C71 A/D
- Various ways to generate external reference voltage (VREF)
- Configuring RA0-RA3 pins

COMMONLY USED A/D TERMINOLOGY

The Ideal Transfer Function

In an A/D converter, an analog voltage is mapped into an N-bit digital value. This mapping function is defined as the transfer function. An ideal transfer is one in which there are no errors or non-linearity. It describes the "ideal" or intended behavior of the A/D. Figure 1 shows the ideal transfer function for the PIC16C7X A/D. Note that the digital output value is 00h for analog input voltage range of 0 to 1LSB. In some converters, the first transition point is at 0.5LSB and not at 1LSB as shown in Figure 2. Either way, knowing the transfer function the user can appropriately interpret the data.

Transition Point

It is the analog input voltage at which the digital output switches from one code to the next. The transition point is typically not a single threshold, rather a small region of uncertainty (see Figure 3) The transition point is therefore defined as the statistical average of many conversions. Stated differently, it is the voltage input at which the uncertainty of the conversion is 50%.

Code Width

It is the distance (voltage differential) between two transition points. Ideally the Code Width should be 1LSB. See Figure 1.

FIGURE 1 - PIC16C7X IDEAL TRANSFER FUNCTION

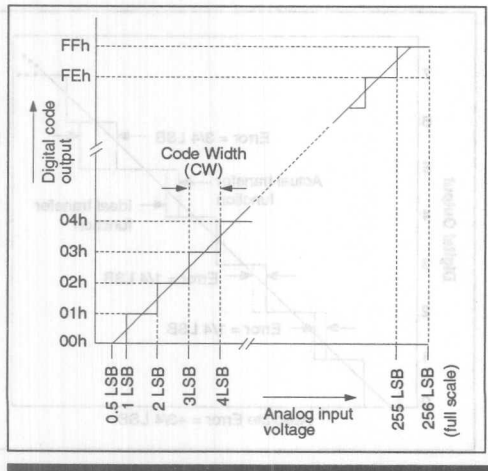
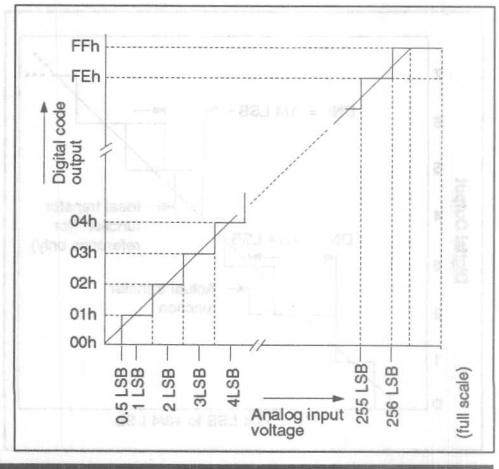


FIGURE 2 - ALTERNATE TRANSFER FUNCTION

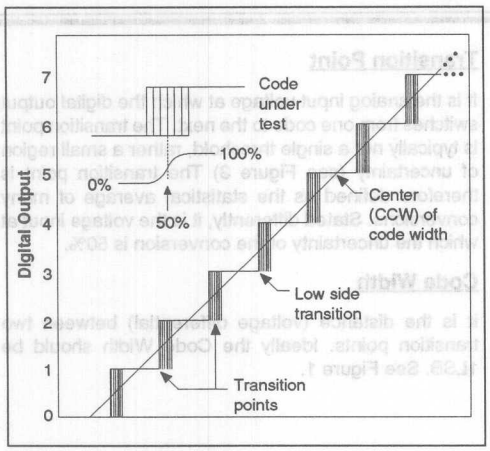


© 1994 Microchip Technology Inc.

Center of Code Width

It is the midpoint between two transition points. See Figure 3.

FIGURE 3 - TRANSITION POINTS



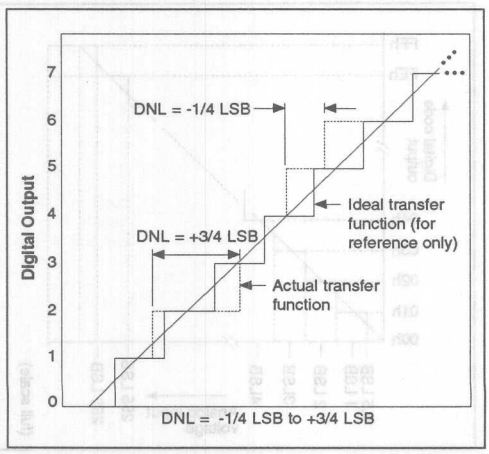
Differential Non-Linearity (DNL)

It is the deviation in code-width from 1LSB (Figure 7). The difference is calculated for each and every transition. The largest difference is reported as DNL.

It is important to note that the DNL is measured after the transfer function is normalized to match offset error and gain error.

Note that the DNL cannot be any less than -1LSB. In the other direction, DNL can be >1LSB.

FIGURE 7 - DIFFERENTIAL NON-LINEARITY



Absolute Error

The maximum deviation between any transition point from the corresponding ideal transfer function is defined as the absolute error. This is how it is measured and reported in the PIC16C7X (Figure 8). The notable difference between absolute error and INL is that the measured data is not normalized for full scale and offset errors.

It is probably the first parameter the user will look at to evaluate an A/D. Sometimes absolute error is reported as the sum of offset, full-scale and integral non linearity errors.

Total Unadjusted Error

It is the same as absolute error. Again, sometimes it is reported as the sum of offset, full-scale and integral non-linearity errors.

No Missing Code

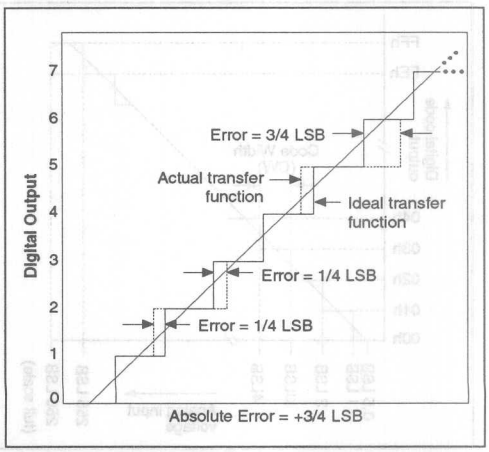
No missing code implies that as the analog input voltage is gradually increased from zero to full scale (or vice versa), all digital codes are produced. Stated otherwise, changing analog input voltage from one quantum of the analog range to the next adjacent range will not produce a change in the digital output by more than one code count.

Monotonic

Monotonicity guarantees that an increase (or decrease) in the analog input value will result in an equal or greater digital code (or less). Monotonicity does not guarantee that there are no missing codes. However, it is an important criterion for feedback control systems. Non-monotonicity may cause oscillations in such a system.

The first derivative of a monotonic function always has the same sign.

FIGURE 8 - ABSOLUTE ERROR



© 1994 Microchip Technology Inc.

Ratiometric Conversion

It is the A/D conversion process where the binary result is a ratio of the supply voltage or reference voltage, the latter being equal to full-scale value by default. The PIC16C7X is a ratiometric A/D converter where the result depends on VDD or VREF.

In some A/D's, an absolute reference is provided resulting in "absolute conversion".

Sample and Hold

In sample and hold type A/D converters, the analog input has a switch (typically a FET switch in CMOS) which is opened for a short duration to capture the analog input voltage onto an on-chip capacitor. Conversion is typically started after the sampling switch is closed.

Track and Hold

It is basically the same as sample and hold, except the sampling switch is typically left on. Therefore the voltage on the on-chip holding capacitor "tracks" the analog input voltage. To begin a conversion, the sampling switch is shut off.

The PIC16C7X A/D falls in this category.

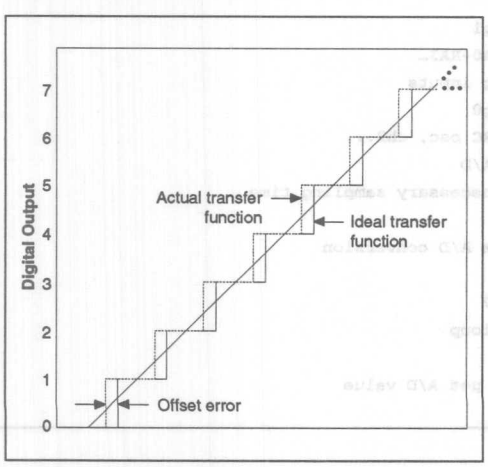
Sampling Time

It is the time required to charge the on-chip holding capacitor to the same value as on the analog input pin. The sampling time depends on the magnitude of the holding capacitor and the source impedance of the analog voltage input.

Offset Error (or Zero Error)

It is the difference between the first actual (measured) transition point and the first ideal transition point as shown in Figure 4. It can be corrected by the user by subtracting the offset error from each conversion result.

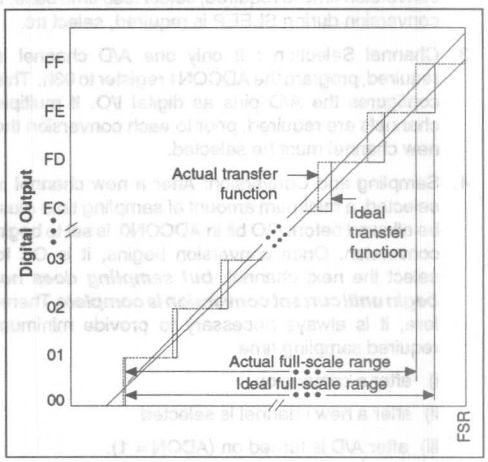
FIGURE 4 - OFFSET ERROR



Full Scale Error (or Gain Error)

It is the difference between the ideal Full Scale and the actual (measured) full scale range (see Figure 5). It is also called gain error, because the error changes the slope of the ideal transfer function creating a gain factor. It can be corrected by the user by multiplying each conversion result by the inverse of the gain.

FIGURE 5 - FULL SCALE ERROR

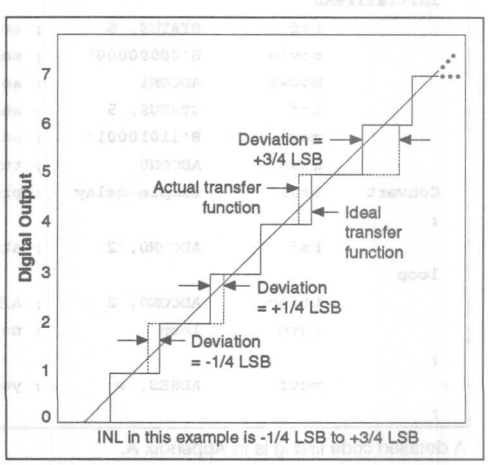


Integral Non-Linearity (INL), or Relative Error

It is the deviation of a transition point from its corresponding point on the ideal transfer curve (Figure 6). The maximum difference is reported as the INL of the converter.

It is important to note that Full Scale Error and the Offset Error are normalized to match end transition points before measuring the INL.

FIGURE 6 - INTEGRAL NON-LINEARITY



© 1994 Microchip Technology Inc.

HOW TO USE THE PIC16C71 A/D

The A/D in the PIC16C71 is easy to set up and use. There are a few considerations:

1. Select either VDD or VREF as reference voltage. More on using VREF input later.
2. Select A/D conversion clock (tad): 2 tosc, 8 tosc, 32 tosc or trc (internal RC clock). For the first three options, make sure that tad ≥ 2.0 µs. If deterministic conversion time is required, select tosc time base. If conversion during SLEEP is required, select trc.
3. Channel Selection : If only one A/D channel is required, program the ADCON1 register to 03h. This configures the A/D pins as digital I/O. If multiple channels are required, prior to each conversion the new channel must be selected.
4. Sampling and Conversion: After a new channel is selected, a minimum amount of sampling time must be allowed before GO bit in ADCON0 is set to begin conversion. Once conversion begins, it is OK to select the next channel, ***but sampling does not begin until current conversion is complete.*** Therefore, it is always necessary to provide minimum required sampling time
   i) after a conversion
   ii) after a new channel is selected
   iii) after A/D is turned on (ADON = 1).
5. Reading Result: Completion of conversion can be determined by either polling GO/$\overline{\text{DONE}}$ bit to cleared, polling the ADIF bit to be set, or waiting for an ADIF interrupt.

ADDITIONAL TIPS:

1. The GO bit and the ADON bit may not be set at once. After the A/D is turned on by setting ADON, at least 5µs time must be allowed before conversion begins, longer if sampling time requirement is not met within 5µs.
2. Aborting a conversion: A conversion can be aborted by clearing GO bit. The A/D converter will stop conversion and revert back to sampling state.
3. Using ADRES register as a normal register: The A/D only writes to ADRES at the end of a conversion. Therefore, it is possible to use ADRES as a normal file register between conversions and when A/D is off.

The following are a few examples of using the A/D.

EXAMPLE 1: HOW TO DO A SIMPLE ADC CONVERSION

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Always ch2, internal RC OSC.
InitializeAD
            bsf         STATUS, 5       ; select pg1
            movlw       B'00000000'     ; select RA0-RA3…
            movwf       ADCON1          ; as analog inputs
            bcf         STATUS, 5       ; select pg0
            movlw       B'11010001'     ; select: RC osc, ch2...
            movwf       ADCON0          ; turn on A/D
Convert     call        sample-delay    ; provide necessary sampling time
;
            bsf         ADCON0, 2       ; start new A/D conversion
loop
            btfsc       ADCON0, 2       ; A/D over?
            goto        loop            ; no then loop
;
            movf        ADRES, w        ; yes then get A/D value
;
```

A detailed code listing is in Appendix A.

© 1994 Microchip Technology Inc.

EXAMPLE 2: HOW TO DO SEQUENTIAL CHANNEL CONVERSIONS

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3 in a round robin fashion, internal RC OSC.
; Load results in 4 consecutive addresses starting at ADTABLE (10h)
;
InitializeAD
          bsf       STATUS, 5     ; select pg1
          movlw     B'00000000'   ; select RA0-RA3…
          movwf     ADCON1        ; as analog inputs
          bcf       STATUS, 5     ; select pg0
          movlw     B'11000001'   ; select: RC osc, ch0...
          movwf     ADCON0        ; turn on A/D
          movlw     ADTABLE       ; point fsr to top of...
          movwf     FSR           ; table
;
new_ad    call      sample_delay  ; provide necessary sampling time
          bsf       ADCON0, 2     ; start new A/D conversion
loop
          btfsc     ADCON0, 2     ; A/D over?
          goto      loop          ; no then loop
;
          movf      adres, w      ; yes then get A/D value
          movwf     0             ; load indirectly
          movlw     4             ; select next channel
          addwd     ADCON0        ;      /
          bcf       ADCON0, 5     ; reset carry over bit.
; increment pointer to correct table offset.
          clrf      temp          ; clear temp register
          btfsc     ADCON0, 3     ; test lsb of channel select
          bsf       temp, 0       ; set if ch1 selected
          btfsc     ADCON0, 4     ; test msb of channel select
          bsf       temp, 1       ;      /
          movlw     ADTABLE       ; get table address
          addwf     temp, w       ; add with temp
          movwf     FSR           ; move into indirect
          goto      new_ad
;
```

A detailed code listing is in Appendix B.

© 1994 Microchip Technology Inc.

EXAMPLE 3: HOW TO WRITE THE INTERRUPT HANDLER FOR THE ADC

```
            org         0x00
            goto        start
            org         0x04
            goto        service_ad      ; interrupt vector
;
;           org         0x10
start
            movlw       B'00000000'     ;init I/O ports
            movwf       PORT_B
            tris        PORT_B
;
            call        InitializeAD
update
            bcf         flag,adover     ; reset software A/D flag
            call        SetupDelay      ; setup delay >= 10uS.
            bcf         ADCON0,adif     ; reset A/D int flag (ADIF
            bsf         ADCON0,adgo     ; start new A/D conversion
            bsf         INTCON,gie      ; enable global interrupt
loop
            btfsc       flag,adover     ; A/D over?
            goto        update          ; yes start new conv.
            goto        loop            ; no then keep checking
; InitializeAD, initializes and sets up the A/D hardware.
; select ch0 to ch3, RC OSC., a/d interrupt.
InitializeAD
            bsf         STATUS, 5       ; select pg1
            movlw       B'00000000'     ; select RA0-RA3...
            movwf       ADCON1          ; as analog inputs
            bcf         STATUS, 5       ; select pg0
            clrf        INTCON          ; clr all interrupts
            bsf         INTCON, 6       ; enable A/D int.
            movlw       B'11010001'     ; select: RC osc, ch2...
            movwf       ADCON0          ; turn on A/D
            return
;
service_ad
            btfss       ADCON0, 1       ; A/D interrupt?
            retfie                      ; no then ignore
            movf        ADRES, W        ; get A/D value
            return                      ; do not enable int
;
```

A detailed code listing is in Appendix C.

© 1994 Microchip Technology Inc.

EXAMPLE 4: HOW TO DO CONVERSIONS DURING SLEEP MODE

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3, internal RC OSC.
; While doing the conversion put unit to sleep.  This will
; minimize digital noise interference.
; Note that ad's RC osc. has to be selected in this instance.
;
InitializeAD
          bsf       STATUS, 5       ; select pg1
          movlw     B'00000000'     ; select RA0-RA3...
          movwf     ADCON1          ; as analog inputs
          bcf       STATUS, 5       ; select pg0
          movlw     B'11000001'     ; select: RC osc, ch0...
          movwf     ADCON0          ; turn on A/D & ADIE
          movlw     ADTABLE         ; point fsr to top of...
          movwf     FSR             ; table
;
new_ad
          bsf       ADCON0, 2       ; start new A/D conversion
          sleep                     ; goto sleep
; when A/D is over program will continue from here
;
          movf      ADRES, w        ;get A/D value
;
```

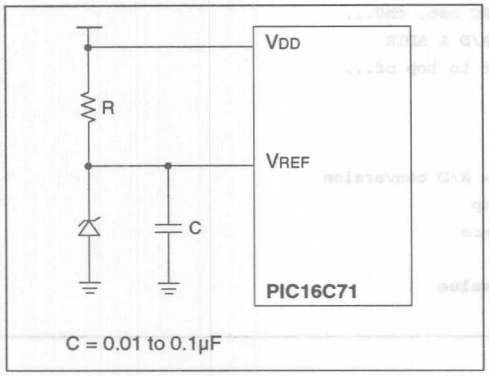
A detailed code listing is in Appendix D.

USING EXTERNAL REFERENCE VOLTAGE

When using external reference voltage, keep in mind that any analog input voltage must not exceed VREF.

An inexpensive way to generate VREF is by employing zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as 10 µA. However, larger currents (1mA - 20mA) are recommended for stability, as well as lower impedance of the VREF source.

FIGURE 9 - LOW COST VOLTAGE REFERENCE



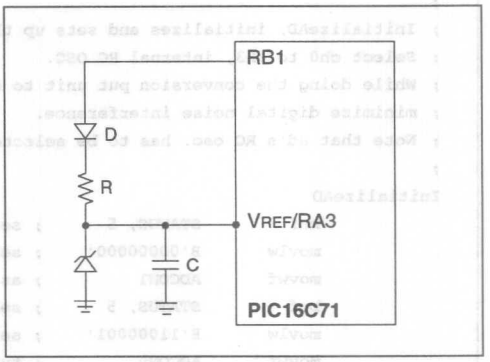
POWER MANAGEMENT IN USING VREF

In power sensitive applications, user may turn on VREF generator using another I/O pin as shown in Figure 10. Drive a "1" on RB1 pin in this example when using the A/D. Drive a "0" on RB1 pin when not using the A/D converter.

Note that this way RB1 is not floating. Even if VREF decays to some intermediate voltage, it will not cause the input buffer on RB1 to draw current.

Alternately, use RA0, RA1 or RA2 pin to supply the current instead of RB1. Configure the RA pin as analog (this will turn off its input buffer). Then use it as a digital output (Figure 11).

FIGURE 10 - POWER-SENSITIVE APPLICATIONS #1



ZENERS AND REFERENCE GENERATORS

Finally, various reference voltage generator chips (typically using on-chip band-gap reference) are available. These are more accurate.

TABLE 1 - ZENERS AND REFERENCE GENERATORS

Zeners	Vz	Tolerance
1N746	3.3V	±5%
1N747	3.6V	±5%
1N748	3.9V	±5%
1N749	4.3V	±5%
1N750	4.7V	±5%
1N751	5.1V	±5%
1N752	5.6V	±5%
Voltage References	**VREF**	**Tolerance**
AD580 (Maxim)	2.5V	±3% to ±0.4%
LM385	2.5V	±1.5%
LM1004	2.5V	±1.2%
LT1009 (LIN. Tech.)	2.5V	±0.2%
LT1019 (LIN. Tech.)	5.0V	±0.2%
LT1021 (LIN. Tech.)	5.0V	±0.05% to ±1%
LT1029 (LIN. Tech.)	5.0V	±0.2% to ±1%

© 1994 Microchip Technology Inc.

VREF IMPEDANCE AND CURRENT SUPPLY REQUIREMENTS

Ideally, VREF should have as low a source impedance as possible. Referring to Figure 9, VREF source impedence ≈ R. However, smaller R increases current consumption. Since VREF is used to charge capacitor arrays inside the A/D converter and the holding capacitor CHOLD ≈ 51pF, the following guideline should be met:

tad = 6 (1K + R) 51.2 pF + 1.677µs

tad = conversion clock. For tad = 2µs and for CHOLD = 50 pF, RVREF ≈ 50Ω.

For VREF impedance higher than this, the conversion clock (tad) should be increased appropriately.

FIGURE 11 - POWER-SENSITIVE APPLICATIONS #2

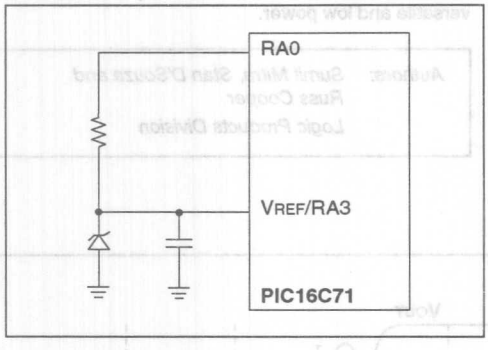


Table 2 gives examples of the maximum rate of conversion per bit, relating to the voltage reference impedance.

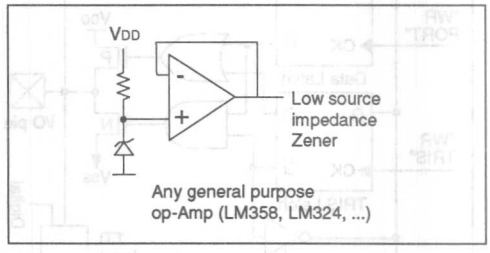
TABLE 2 - MAXIMUM RATE OF CONVERSION / BIT

RVREF	Tad (Max)
1K	2.29 µs
5K	3.52 µs
10K	5.056 µs
50K	16.66 µs
100K	32.70 µs

Assumes no external capacitors.

To achieve a low source impedance when using a Zener diode, a voltage follower circuit is recommended. This is shown in Figure 11A.

FIGURE 11A - VOLTAGE FOLLOWER CIRCUIT



CONFIGURING PORT A INPUTS AS ANALOG OR DIGITAL

Two bits in ADCON1 register PCFG1 and PCFG0 control how pins RA0–RA3 are configured. When any of these pins are selected as analog:

- The digital input buffer is turned off to save current (see Figure 12). Reading the port will read this pin as '0'.
- TRIS bit still controls the output buffer on this pin. So, normally the TRIS bit will be set (input).
- However, if the TRIS bit is cleared, then the pin will output whatever is in the data latch.

When any of these pins are selected as digital:

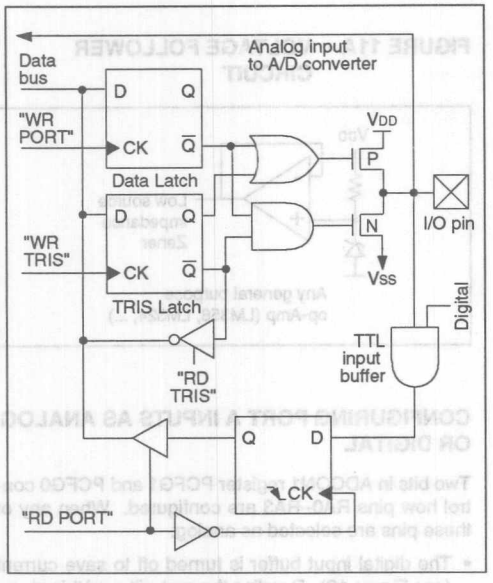
- The analog input still directly connects to the A/D and therefore the pin can be used as analog input.
- The digital input buffer is not disabled.

The user has, therefore, great flexibility in configuring these pins.

3

© 1994 Microchip Technology Inc.

FIGURE 12 - BLOCK DIAGRAM OF RA0-RA3 PINS



CURRENT CONSUMPTION THROUGH INPUT BUFFER

A CMOS input buffer will draw current when the input voltage is around its threshold. (See Figure 13.)

In power-sensitive applications, the RA pins when used as analog inputs should be configured as "analog" to avoid unintended power drain.

Other considerations and tips:

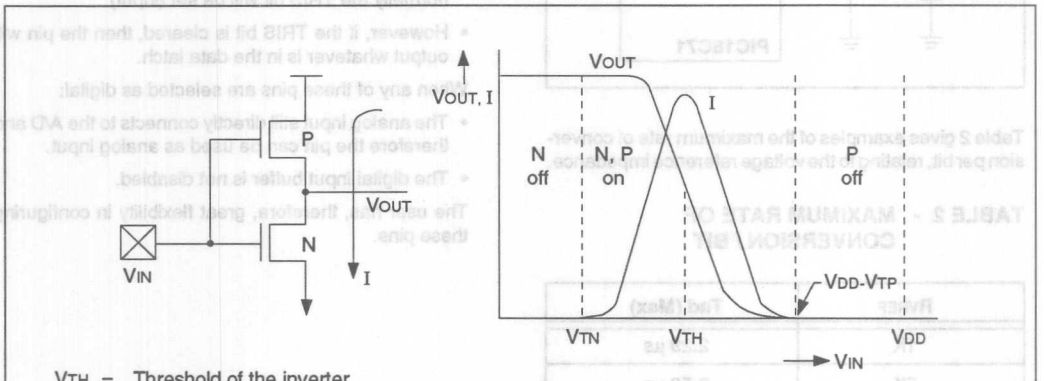
1. If possible, avoid any digital output next to analog inputs.
2. Avoid digital inputs that switch frequently (e.g., clocks) next to analog inputs.
3. If VREF is used, then no analog pin being sampled should exceed VREF.

SUMMARY

The PIC16C71 A/D converter is simple to use. It is versatile and low power.

Authors: *Sumit Mitra, Stan D'Souza and Russ Cooper*
Logic Products Division

FIGURE 13 - A SIMPLE CMOS INPUT BUFFER:



VTH = Threshold of the inverter
VTN = Device threshold of NMOS pull-down
-VTP = Device threshold of PMOS pull-up
I = On-current (or through current) of the inverter
IMAX = Maximum on-current occurs when VIN = VTH. Value of IMAX depends on the sizes of the devices. The larger the devices, the faster the input buffer, and the larger the value of IMAX. Typically, IMAX ≈ 0.2mA – 1mA.

© 1994 Microchip Technology Inc.

APPENDIX A - SINGLE CHANNEL A/D (SAD)

```
MPASM 1.00 Released       SAD.ASM   7-15-1994  13:27:21          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;TITLE   "Single channel A/D (SAD)"
                     0002 ;This program is a simple implementation of the PIC16C71's
                     0003 ;A/D. 1 Channel is selected (CH0).
                     0004 ;The A/D is configured as follows:
                     0005 ;       Vref = +5V internal.
                     0006 ;       A/D Osc. =  internal RC
                     0007 ;       A/D Channel = CH0
                     0008 ;Hardware for this program is the PICDEMO board.
                     0009 ;
                     0010 ;
                     0011         LIST P=16C71,F=INHX8M
                     0012 ;
                     0013     include "picreg.equ"
                     0083
                     0013
                     0014 ;
0010                 0015 TEMP    EQU     10h
0001                 0016 adif    equ     1
0002                 0017 adgo    equ     2
                     0018 ;
                     0019         ORG     0x00
                     0020 ;
                     0021 ;
0000 2810            0022         goto    start
                     0023 ;
                     0024         org     0x04
0004 281C            0025         goto    service_int      ;interrupt vector
                     0026 ;
                     0027 ;
                     0028         org     0x10
                     0029 start
0010 3000            0030         movlw   B'00000000'      ;set port b as
0011 0086            0031         movwf   PORT_B           ;all outputs
0012 0066            0032         tris    PORT_B           ;       /
                     0033 ;
0013 201D            0034         call    InitializeAD
                     0035 update
0014 0809            0036         movf    ADRES,W          ;get a/d value
0015 0086            0037         movwf   PORT_B           ;output to port b
0016 2025            0038         call    SetupDelay       ;setup time >= 10uS.
0017 1088            0039         bcf     ADCON0,adif      ;clear int flag
0018 1508            0040         bsf     ADCON0,adgo      ;start new conversion
                     0041 loop
0019 1888            0042         btfsc   ADCON0,adif      ;a/d done?
001A 2814            0043         goto    update           ;yes then update new value.
001B 2819            0044         goto    loop             ;no then keep checking
                     0045 ;
                     0046 ;no interrupts are enabled, so if the program ever reaches here,
                     0047 ;it should be returned with the global interrupts disabled.
                     0048 service_int
001C 0008            0049         return                   ;do not enable global.
                     0050 ;
                     0051 ;
                     0052 ;
                     0053 ;InitializeAD, initializes and sets up the A/D hardware.
                     0054 ;Select ch0 to ch3 as analog inputs, fosc/2 and read ch3.
                     0055 ;
                     0056 InitializeAD
001D 1683            0057         bsf     STATUS,5         ;select pg1
001E 3000            0058         movlw   B'00000000'      ;select ch0-ch3...
001F 0088            0059         movwf   ADCON1           ;as analog inputs
```

© 1994 Microchip Technology Inc.

```
0020 1283           0060         bcf     STATUS,5        ;select pg0
0021 30C1           0061         movlw   B'11000001'     ;select:RC,ch0..
0022 0088           0062         movwf   ADCON0          ;turn on A/D.
0023 0189           0063         clrf    ADRES           ;clr result reg.
0024 0008           0064         return
                    0065 ;
                    0066 ;This routine is a software delay of 10uS for the a/d setup.
                    0067 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0068 ;a value of 3 to give 9uS, plus the move etc should result in
                    0069 ;a total time of > 10uS.
                    0070 SetupDelay
0025 3003           0071         movlw   .3
0026 0090           0072         movwf   TEMP
                    0073 SD
0027 0B90           0074         decfsz  TEMP
0028 2827           0075         goto    SD
0029 0008           0076         return
                    0077
                    0078
                    0079         END
                    0080
                    0081


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X----- XXXXXXXXXXXXXXXX XXXXXXXXXX---- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX B

```
MPASM 1.00 Released      SLPAD.ASM   7-15-1994  13:27:15                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001
                     0002 ;TITLE   "A/D in Sleep Mode"
                     0003 ;This program is a simple implementation of the PIC16C71's
                     0004 ;A/D feature. This program demonstrates
                     0005 ;how to do a a/d in sleep mode on the PIC16C71.
                     0006 ;The A/D is configured as follows:
                     0007 ;       Vref = +5V internal.
                     0008 ;       A/D Osc. = internal RC
                     0009 ;       A/D Interrupt = OFF
                     0010 ;       A/D Channels = ch 0
                     0011 ;
                     0012 ;The ch0 A/D result is displayed as a 8 bit binary value
                     0013 ;on 8 leds connected to port b. Hardware used is that of
                     0014 ;the PICDEM0 board.
                     0015 ;
                     0016 ;
                     0017         LIST P=16C71,F=INHX8M
                     0018 ;
                     0019     include "picreg.equ"
                     0083
                     0019
                     0020 ;
0010                 0021 TEMP    EQU     10h
0001                 0022 adif    equ     1
0002                 0023 adgo    equ     2
                     0024 ;
                     0025 ;
                     0026         ORG     0x00
                     0027 ;
                     0028 ;
0000 2810            0029         goto    start
                     0030 ;
                     0031         org     0x04
0004 281B            0032         goto    service_int      ;interrupt vector
                     0033 ;
                     0034 ;
                     0035         org     0x10
                     0036 start
0010 3000            0037         movlw   B'00000000'      ;make port b all
0011 0086            0038         movwf   PORT_B           ;outputs.
0012 0066            0039         tris    PORT_B           ;       /
                     0040 ;
0013 201C            0041         call    InitializeAD
                     0042 update
0014 0809            0043         movf    ADRES,W
0015 0086            0044         movwf   PORT_B           ;save in table
0016 2025            0045         call    SetupDelay       ;
0017 1088            0046         bcf     ADCON0,adif      ;clr a/d flag
0018 1508            0047         bsf     ADCON0,adgo      ;start new a/d conversion
                     0048 ;
0019 0063            0049         sleep
001A 2814            0050         goto    update           ;wake up and update
                     0051 ;

                     0052 service_int
001B 0008            0053         return                   ;do not enable int
                     0054 ;
                     0055 ;InitializeAD, initializes and sets up the A/D hardware.
                     0056 InitializeAD
001C 1683            0057         bsf     STATUS,5         ;select pg1
001D 3000            0058         movlw   B'00000000'      ;select ch0-ch3...
```

© 1994 Microchip Technology Inc.

```
001E 0088           0059          movwf   ADCON1          ;as analog inputs
001F 1283           0060          bcf     STATUS,5        ;select pg0
0020 30C1           0061          movlw   B'11000001'     ;select:internal RC, ch0.
0021 0088           0062          movwf   ADCON0          ;turn on a/d
0022 018B           0063          clrf    INTCON          ;clear all interrupts
0023 170B           0064          bsf     INTCON,ADIE     ;enable a/d
0024 0008           0065          return
                    0066 ;
                    0067 ;This routine is a software delay of 10uS for the a/d setup.
                    0068 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0069 ;a value of 3 to give 9uS, plus the move etc should result in
                    0070 ;a total time of > 10uS.
                    0071 SetupDelay
0025 3003           0072          movlw   .3
0026 0090           0073          movwf   TEMP
                    0074 SD
0027 0B90           0075          decfsz  TEMP
0028 2827           0076          goto    SD
0029 0008           0077          return
                    0078
                    0079 ;
                    0080
                    0081          END
                    0082
                    0083


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X----- XXXXXXXXXXXXXXXX XXXXXXXXXX---- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX C

```
MPASM 1.00 Released      INTAD.ASM   7-15-1994  13:27:32          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001
                     0002 ;TITLE       "Single channel A/D with interrupts"
                     0003 ;This program is a simple implementation of the PIC16C71's
                     0004 ;A/D. 1 Channel is selected (CH0). A/D interrupt is turned on,
                     0005 ;hence on completion of a/d conversion, an interrupt is generated.
                     0006 ;The A/D is configured as follows:
                     0007 ;       Vref = +5V internal.
                     0008 ;       A/D Osc. = internal RC Osc.
                     0009 ;       A/D Interrupt = On
                     0010 ;       A/D Channel = CH0
                     0011 ;
                     0012 ;The A/D result is displayed as a 8 bit value on 8 leds connected
                     0013 ;to port b. Hardware setup is the PICDEMO board.
                     0014 ;
                     0015 ;
                     0016         LIST P=16C71,F=INHX8M
                     0017 ;
                     0018     include "picreg.equ"
                     0083
                     0018
                     0019 ;
0010                 0020 flag    equ     10
0011                 0021 TEMP    equ     11
0000                 0022 adover  equ     0
0001                 0023 adif    equ     1
0002                 0024 adgo    equ     2
0006                 0025 adie    equ     6
0007                 0026 gie     equ     7
0005                 0027 rp0     equ     5
                     0028 ;
                     0029         ORG     0x00
                     0030 ;
                     0031 ;
0000 2810            0032         goto    start
                     0033 ;
                     0034         org     0x04
0004 281C            0035         goto    service_ad      ;interrupt vector
                     0036 ;
                     0037 ;
                     0038         org     0x10
                     0039 start
0010 3000            0040         movlw   B'00000000'     ;init i/o ports
0011 0086            0041         movwf   PORT_B
0012 0066            0042         tris    PORT_B
                     0043 ;
0013 2022            0044         call    InitializeAD
                     0045 update
0014 1010            0046         bcf     flag,adover     ;reset software a/d flag
0015 202B            0047         call    SetupDelay      ;setup delay >= 10uS.
0016 1088            0048         bcf     ADCON0,adif     ;reset a/d int flag (ADIF)
0017 1508            0049         bsf     ADCON0,adgo     ;start new a/d conversion
0018 178B            0050         bsf     INTCON,gie      ;enable global interrupt
                     0051 loop

0019 1810            0052         btfsc   flag,adover     ;a/d over?
001A 2814            0053         goto    update          ;yes start new conv.
001B 2819            0054         goto    loop            ;no then keep checking
                     0055 ;
                     0056 service_ad
001C 1C88            0057         btfss   ADCON0,adif     ;ad interrupt?
001D 0009            0058         retfie                  ;no then ignore
001E 0809            0059         movf    ADRES,W         ;get a/d value
```

© 1994 Microchip Technology Inc.

```
001F 0086           0060        movwf   PORT_B          ;output to port b
0020 1410           0061        bsf     flag,adover     ;a/d done set
0021 0008           0062        return                  ;do not enable int
                    0063 ;
                    0064 ;
                    0065 ;InitializeAD, initializes and sets up the A/D hardware.
                    0066 ;select ch0 to ch3, RC OSC., a/d interrupt.
                    0067 InitializeAD
0022 1683           0068        bsf     STATUS,rp0      ;select pg1
0023 3000           0069        movlw   B'00000000'     ;select ch0-ch3...
0024 0088           0070        movwf   ADCON1          ;as analog inputs
0025 1283           0071        bcf     STATUS,rp0      ;select pg0
0026 018B           0072        clrf    INTCON          ;clr all interrupts
0027 170B           0073        bsf     INTCON,adie     ;enable a/d int.
0028 30C1           0074        movlw   B'11000001'     ;select:RC osc,ch0...
0029 0088           0075        movwf   ADCON0          ;turn on a/d
002A 0008           0076        return
                    0077 ;
                    0078 ;This routine is a software delay of 10uS for the a/d setup.
                    0079 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0080 ;a value of 3 to give 9uS, plus the move etc should result in
                    0081 ;a total time of > 10uS.
                    0082 SetupDelay
002B 3003           0083        movlw   .3
002C 0091           0084        movwf   TEMP
                    0085 SD
002D 0B91           0086        decfsz  TEMP
002E 282D           0087        goto    SD
002F 0008           0088        return
                    0089 ;
                    0090 ;
                    0091        END
                    0092
                    0093


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X----- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX D

```
MPASM 1.00 Released    MULTAD.ASM   7-15-1994  13:27:26          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;TITLE   "A/D using Multiple Channels"
                     0002 ;This program is a simple implementation of the PIC16C71's
                     0003 ;A/D feature. This program demonstrates
                     0004 ;how to select multiple channels on the PIC16C71.
                     0005 ;The A/D is configured as follows:
                     0006 ;       Vref = +5V internal.
                     0007 ;       A/D Osc. = internal RC osc.
                     0008 ;       A/D Interrupt = Off
                     0009 ;       A/D Channels = all in a "Round Robin" format.
                     0010 ;       A/D reuslts are stored in ram locations as follows:
                     0011 ;       ch0 -> ADTABLE + 0
                     0012 ;       ch1 -> ADTABLE + 1
                     0013 ;       ch2 -> ADTABLE + 2
                     0014 ;       ch3 -> ADTABLE + 3
                     0015 ;
                     0016 ;The ch0 A/D result is displayed as a 8 bit value on 8 leds
                     0017 ;connected to port b.
                     0018 ;Hardware: PICDEM0 board.
                     0019 ;                Stan D'Souza 7/6/93.
                     0020 ;
                     0021         LIST P=16C71,F=INHX8M
                     0022 ;
                     0023    include "picreg.equ"
                     0083
                     0023
                     0024 ;
0010                 0025 TEMP    EQU     10h
0001                 0026 adif    equ     1
0002                 0027 adgo    equ     2
                     0028 ;
0006                 0029 ch2     equ     6
0007                 0030 ch3     equ     7
000C                 0031 flag    equ     0C
0020                 0032 ADTABLE equ     20
                     0033 ;
                     0034         ORG     0x00
                     0035 ;
                     0036 ;
0000 2810            0037         goto    start
                     0038 ;
                     0039         org     0x04
0004 2823            0040         goto    service_int     ;interrupt vector
                     0041 ;
                     0042 ;
                     0043         org     0x10
                     0044 start
0010 3000            0045         movlw   B'00000000'     ;make port b
0011 0086            0046         movwf   PORT_B          ;as all outputs
0012 0066            0047         tris    PORT_B          ;       /
                     0048 ;
0013 2024            0049         call    InitializeAD
                     0050 update
0014 0809            0051         movf    ADRES,W

0015 0080            0052         movwf   0               ;save in table
0016 3020            0053         movlw   ADTABLE         ;chk if ch0
0017 0204            0054         subwf   FSR,W           ;       /
0018 1D03            0055         btfss   STATUS,Z        ;yes then skip
0019 281C            0056         goto    NextAd          ;else do next channel
001A 0809            0057         movf    ADRES,W         ;get a/d value
001B 0086            0058         movwf   PORT_B          ;output to port b
                     0059 NextAd
```

```
001C 202E            0060          call    NextChannel     ;select next channel
001D 203A            0061          call    SetupDelay      ;set up > = 10uS
001E 1088            0062          bcf     ADCON0,adif     ;clear flag
001F 1508            0063          bsf     ADCON0,adgo     ;start new a/d conversion
                     0064 loop
0020 1888            0065          btfsc   ADCON0,adif     ;a/d done?
0021 2814            0066          goto    update          ;yes then update
0022 2820            0067          goto    loop            ;wait till done
                     0068 ;
                     0069 service_int
0023 0008            0070          return                  ;do not enable int
                     0071 ;
                     0072 ;
                     0073 ;InitializeAD, initializes and sets up the A/D hardware.
                     0074 InitializeAD
0024 1683            0075          bsf     STATUS,5        ;select pg1
0025 3000            0076          movlw   B'00000000'     ;select ch0-ch3...
0026 0088            0077          movwf   ADCON1          ;as analog inputs
0027 1283            0078          bcf     STATUS,5        ;select pg0
0028 30C1            0079          movlw   B'11000001'     ;select:fosc/2, ch0.
0029 0088            0080          movwf   ADCON0          ;turn on a/d
002A 3020            0081          movlw   ADTABLE         ;get top of table address
002B 0084            0082          movwf   FSR             ;load into indirect reg
002C 0189            0083          clrf    ADRES           ;clr result reg.
002D 0008            0084          return
                     0085 ;
                     0086 ;NextChannel, selects the next channel to be sampled in a
                     0087 ;"round-robin" format.
                     0088 NextChannel
002E 3008            0089          movlw   0x08            ;get channel offset
002F 0788            0090          addwf   ADCON0          ;add to conf. reg.
0030 1288            0091          bcf     ADCON0,5        ;clear any carry over
                     0092 ;increment pointer to correct a/d result register
0031 0190            0093          clrf    TEMP
0032 1988            0094          btfsc   ADCON0,3        ;test lsb of chnl select
0033 1410            0095          bsf     TEMP,0          ;set if ch1 or ch3
0034 1A08            0096          btfsc   ADCON0,4        ;test msb of chnl select
0035 1490            0097          bsf     TEMP,1          ;set if ch0 or ch2
0036 3020            0098          movlw   ADTABLE         ;get top of table
0037 0710            0099          addwf   TEMP,W          ;add with temp
0038 0084            0100          movwf   FSR             ;allocate new address
0039 0008            0101          return
                     0102 ;
                     0103 ;This routine is a software delay of 10uS for the a/d setup.
                     0104 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                     0105 ;a value of 3 to give 9uS, plus the move etc should result in
                     0106 ;a total time of > 10uS.
                     0107 SetupDelay
003A 3003            0108          movlw   .3
003B 0090            0109          movwf   TEMP
                     0110 SD
003C 0B90            0111          decfsz  TEMP
003D 283C            0112          goto    SD
003E 0008            0113          return
                     0114
                     0115 ;
                     0116
                     0117          END
                     0118
                     0119

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X------ XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-
0040 : -------- -------- -------- --------

All other memory blocks unused.

Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN557

Four Channel Digital Voltmeter with Display and Keyboard

INTRODUCTION

The PIC16C71 is a member of a new family of 8-bit microcontrollers, namely the PIC16CXX. The salient features of the PIC16C71 are:

- Improved and enhanced instruction set
- 14-bit instruction word
- Interrupt capability
- On-chip four channel, 8-bit A/D converter

This application note demonstrates the capability of the PIC16C71. To make this application note easier for the end user, it has been broken down into four sub-sections:

Section 1: Implements the multiplexing of four 7 segment LEDs with the PIC16C71.

Section 2: Implements the multiplexing of four 7 segment LEDs as well as the sampling of a 4x4 Keypad.

Section 3: Implements the multiplexing of four 7 segment LEDs as well as the sampling of one A/D channel.

Section 4: Implements the multiplexing of four 7 segment LEDs, sampling a 4x4 keypad and four A/D channels.

IMPLEMENTATION

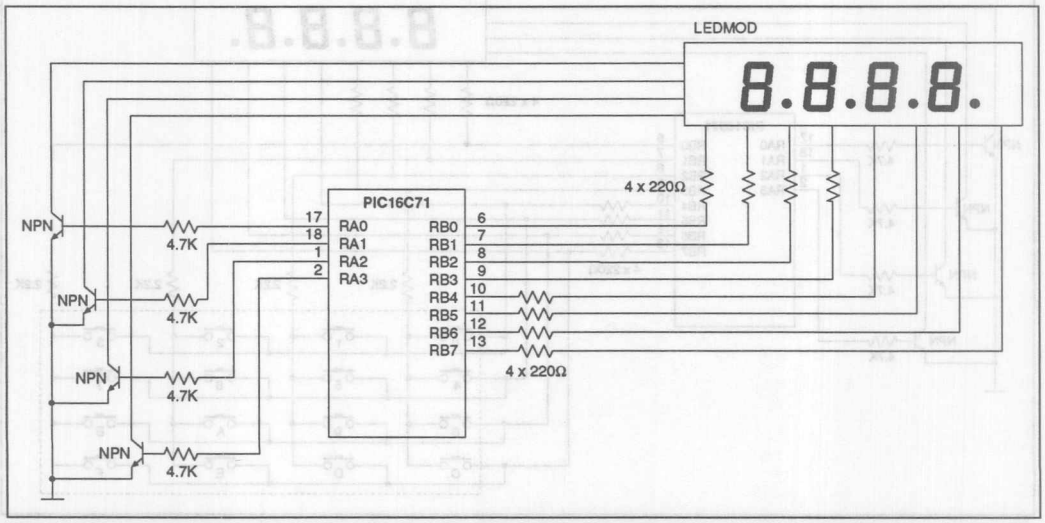
SECTION 1: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS

Hardware

The PIC16C71's I/O ports have an improved sink/source specification. Each I/O pin can sink up to 25 mA and source 20 mA, in addition total Port B source current is 100 mA and sink current is 150 mA. Port A is rated for 50 mA source current and 80 mA sink current. This makes the PIC16C71 ideal for driving 7 segment LEDs. Since the total number of I/O is limited to 13, the 8-bit Port B is used to drive the 8 LEDs, while external sink transistors or MOSFETs are used to sink the digit current (See Figure 1). Another alternative is to use ULN2003 open collector sink current drivers, which are available in 16 pin DIP or very small S0-16 packages. Each transistor on the ULN2003 can sink a maximum of 500 mA and the base drive can be directly driven from the Port A pins.

3

FIGURE 1 - MULTIPLEXING FOUR 7 SEGMENT LEDS



© 1994 Microchip Technology Inc.

Software

The multiplexing is achieved by turning on each LED for a 5 msec duration every 20 µs. This gives an update rate of 50 Hz, which is quite exceptable to the human eye as a steady display. The 5 msec time base is generated by dividing the 4.096 MHz oscillator clock. The internal prescaler is configured to be a divide by 32 and assigned to the RTCC. The RTCC is pre-loaded with a value = 96. The RTCC will increment to FF and then roll over to 00 after a period = (256-96)*(32*4/4096000) = 5 µs. When the RTCC rolls over, the RTIF flag is set and since the RTIE and GIE bits are enabled, an interrupt is generated.

The software implements a simple timer which increments at a 1 second rate. Every second, the 4 nibble (two 8-bit registers MsdTime and LsdTime) are incremented in a BCD format. The lower 4 bits of LsdTime correspond to the least significant digit (LSD) on the display. The high 4 bits of LsdTime correspond to the second significant digit of the display and so on. Depending on which display is turned on, the corresponding 4 bit BCD value is extracted from either MsdTime or LsdTime, and decoded to a 7 segment display. The RTCC interrupt is generated at a steady rate of 5 µs and given an instruction time of 1 us. The entire display update program can reside in the interrupt service routine with no chance of getting an interrupt within an interrupt. The Code listing for Section 1 is in Appendix A.

SECTION 2: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND SCANNING A 4X4 KEYPAD

Hardware

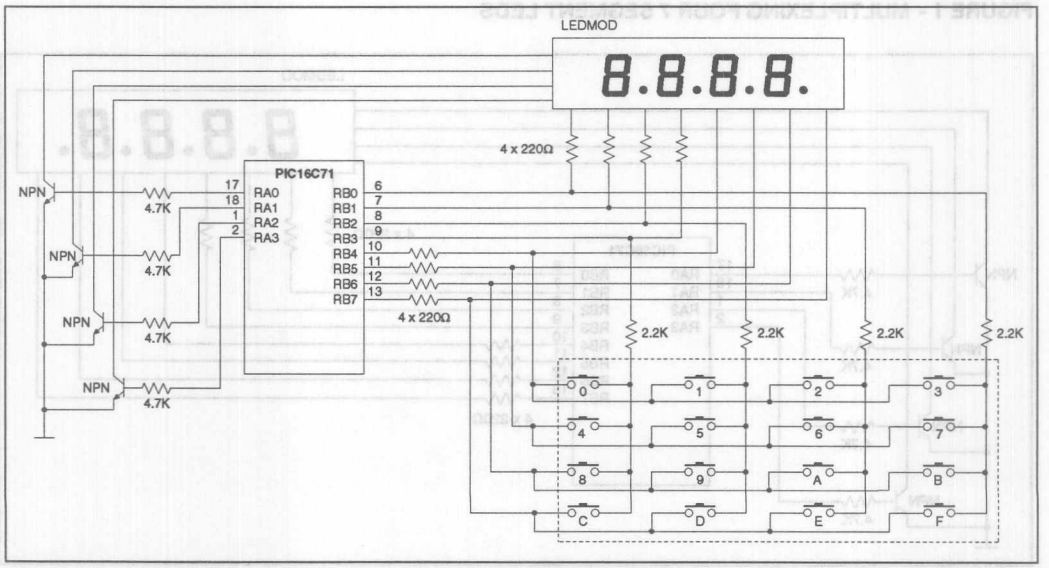
A 4x4 keypad can be very easily interfaced to the PIC16C71's Port B (see Figure 2). Internal pull-ups on pins RB4 to RB7 can be enabled and disabled by setting the RBPU bit in the OPTION register. The internal pull-ups have a value of 20K at 5V (typical). In order to sense a low level at the input, the switch is "connected" to ground through a 2.2K resistor. A key hit normally lasts from 50 msec to as long as a person holds the key down. In order not to miss any key hits, the keypad is sampled every 20 µs (just after the update of the MSD).

Software

To sample the keypad, the digit sinks are first disabled. Port B is then configured with RB4-RB7 as inputs and RB0 -RB3 as outputs driven high. The pull-ups on RB4-RB7 are enabled. Sequentially RB0 to RB3 are made low while RB4 to RB7 are checked for a key hit (a low level). One key hit per scan is demonstrated in this program. Multiple key hits per scan can very easily be implemented. Once the key hit is sensed, a 40 msec debounce period elapses before key sampling is resumed. No more key hits are sensed until the present key is released. This prevents erroneous key inputs.

The program basically inputs the key hit and displays its value as a hexadecimal character on the multiplexed 7-segment LEDs. The Code Listing for Section 2 is in Appendix B.

FIGURE 2 - MULTIPLEXING FOUR 7 SEGMENT LEDS WITH A 4X4 KEYPAD



© 1994 Microchip Technology Inc.

SECTION 3: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND THE A/D CHANNEL 0

Hardware

The four analog channels are connected to RA0-RA3. If any of these pins are used normally as digital I/O, they can momentarily be used as analog inputs. In order to avoid interference from the analog source, it is advisable to buffer the analog input through a voltage follower op amp, however, it is not always necessary. Figure 3A and 3B show some typical configurations. In this application, the analog input is a potentiometer whose wiper is connected through an RC network to channel 0. The RC is necessary in order to smooth out the analog voltage. The RC does contribute to a delay in the sampling time, however the stability of the analog reading is greatly improved.

Software

The analog input is sampled every 20 msec. The digit sinks and the drivers are turned off i.e. Port A is configured as an input and Port B outputs are made low. A 1msec settling time is allowed for the external RC network connected to the analog input to settle and then the A/D conversion is started. The result is read then converted from an 8-bit binary value to a 3-digit BCD value which is then displayed on the 7 Segment LEDs. The Code Listing for Section 3 is in Appendix C.

FIGURE 3A - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT

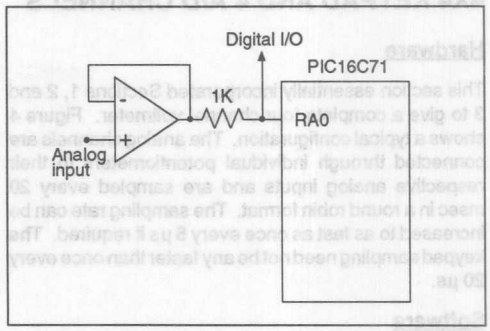
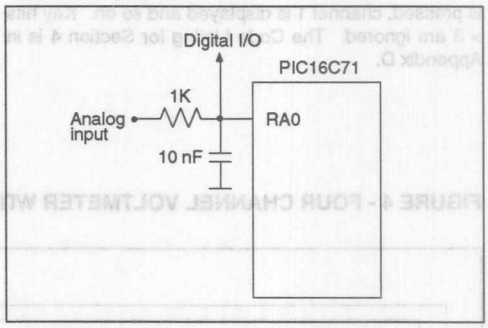
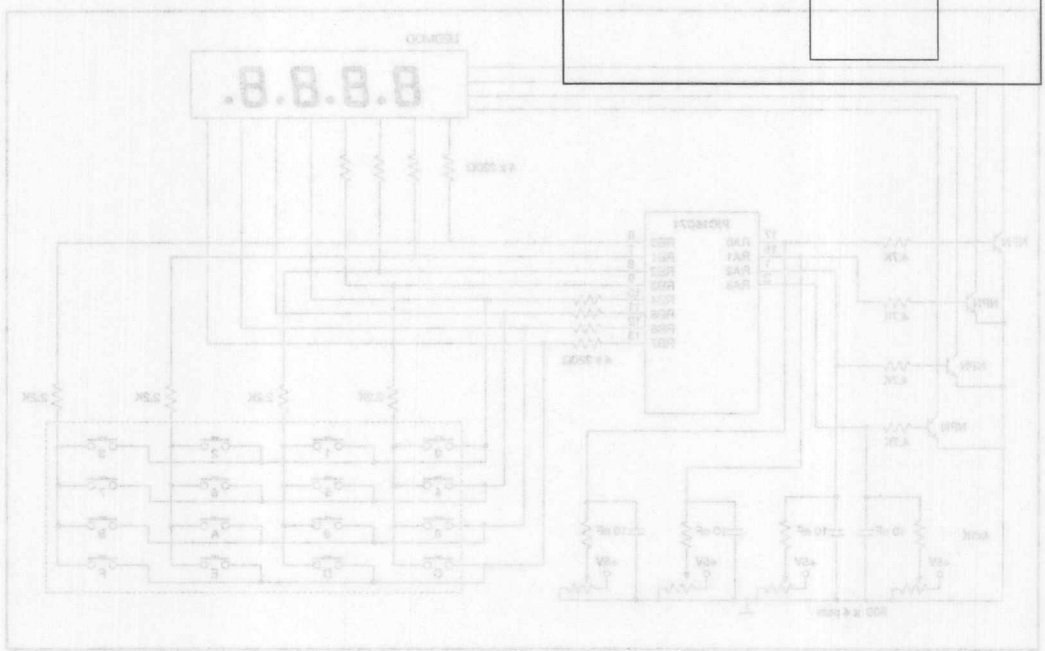


FIGURE 3B - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT





© 1994 Microchip Technology Inc.

SECTION 4: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS WITH A 4X4 KEYPAD AND 4 A/D CHANNELS

Hardware

This section essentially incorporated Sections 1, 2 and 3 to give a complete four channel voltmeter. Figure 4 shows a typical configuration. The analog channels are connected through individual potentiometers to their respective analog inputs and are sampled every 20 msec in a round robin format. The sampling rate can be increased to as fast as once every 5 µs if required. The keypad sampling need not be any faster than once every 20 µs.

Software

The program samples the analog inputs and saves the result in four consecutive locations starting at "ADVALUE", with channel 0 saved at the first location and so on. By default, channel 0 is displayed. If Key 1 is pressed, channel 1 is displayed and so on. Key hits > 3 are ignored. The Code Listing for Section 4 is in Appendix D.

Code Size

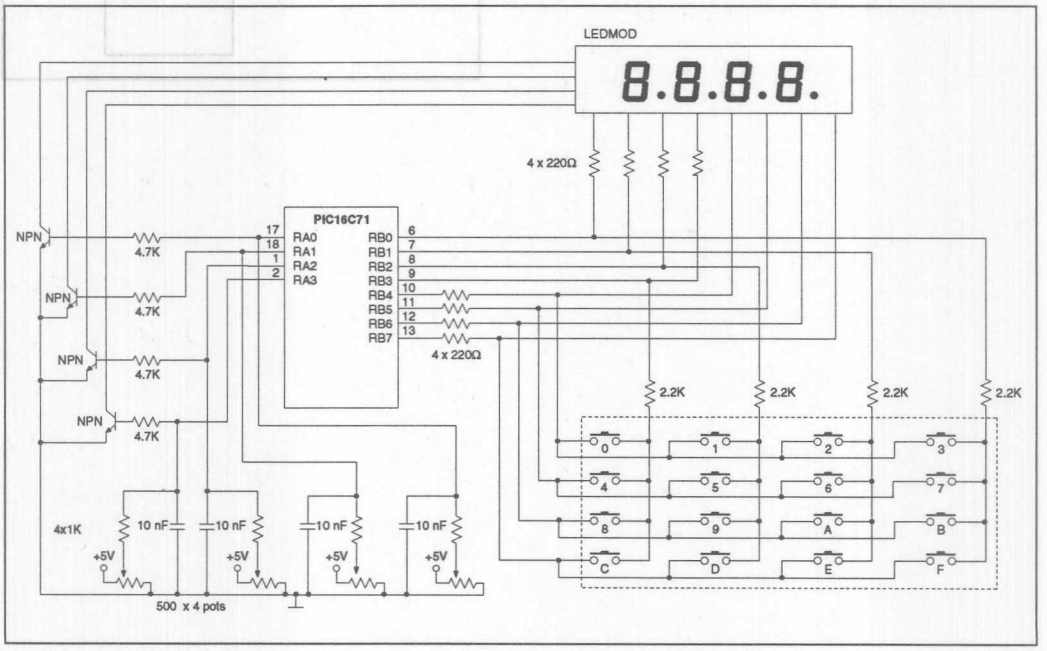
Section 1:	Program Memory:	139
	Data Memory:	6
Section 2:	Program Memory:	207
	Data Memory:	13
Section 3:	Program Memory:	207
	Data Memory:	17
Section 4:	Program Memory:	
	Data Memory:	

Conclusion

The four A/D channels on the PIC16C71 can be multiplexed with digital I/O, thus reducing overall pin counts and improving I/O pin usage in an analog application.

Author: Stan D'Souza, Logic Products Division

FIGURE 4 - FOUR CHANNEL VOLTMETER WITH DISPLAY AND KEYPAD



© 1994 Microchip Technology Inc.

APPENDIX A

```
MPASM B0.50                                              PAGE  1


                        ;*****************************************************************
                        ;This program is to demonstrate how to multiplex four 7 segment LED
                        ;digits using a PIC16C71. The four digits will start at 0000 and
                        ;increment at a 1 sec rate up to 9999.
                        ;The LEDs are updated every 5 msec, for a multiplexing rate of 20 msec.
                        ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                        ;
                        ;                                        Stan D'Souza 5/8/93
                        ;*****************************************************************
                            LIST P=16C71, F=INHX8M
                        ;
                            include "picreg.equ"


                        ;
000C                    TempC       equ     0x0c        ;temp general purpose regs
000D                    TempD       equ     0x0d
000E                    TempE       equ     0x0e
000F                    Count       equ     0x0f        ;count
0010                    MsdTime     equ     0x10        ;most significant Timer
0011                    LsdTime     equ     0x11        ;Least significant Timer
0001                    OptionReg   equ     1
0002                    PCL         equ     2
0026                    BcdMsd      equ     26
0027                    Bcd         equ     27
                        ;
                                    org     0
0000 2805                           goto    Start       ;skip over interrupt vector
                        ;
                                    org     4
0004 281D                           goto    ServiceInterrupts
                        ;
                        Start
0005 2008                           call    InitPorts
0006 2012                           call    InitTimers
                        loop
0007 2807                           goto    loop
                        ;
                        InitPorts
0008 1683                           bsf     STATUS,RP0  ;select pg 1
0009 3003                           movlw   3           ;make RA0-3 digital I/O
000A 0108                           movwf   ADCON1      ;       /
000B 0205                           clrf    TRISA       ;make RA0-4 outputs
000C 0206                           clrf    TRISB       ;make RB0-7 outputs
000D 1283                           bcf     STATUS,RP0  ;select page 0
000E 0185                           clrf    PORT_A      ;make all outputs low
000F 0186                           clrf    PORT_B      ;       /
0010 1585                           bsf     PORT_A,3    ;enable MSB digit sink
0011 0008                           return
                        ;
                        ;
                        ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                        ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                        ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                        ;end result is that we get a rtcc interrupt every 5msec.
                        InitTimers
0012 0190                           clrf    MsdTime     ;clr timers
0013 0191                           clrf    LsdTime     ;       /
0014 1683                           bsf     STATUS,RP0  ;select pg 1
0015 3084                           movlw   B'10000100'  ;assign ps to rtcc
```

© 1994 Microchip Technology Inc.

```
0016 0081                   movwf   OptionReg               ;ps = 32
0017 1283                   bcf     STATUS,RP0              ;select pg 0
0018 3020                   movlw   B'00100000'             ;enable rtcc interrupt
0019 008B                   movwf   INTCON                  ;
001A 3060                   movlw   .96                     ;preload rtcc
001B 0081                   movwf   RTCC                    ;start counter
001C 0009                   retfie
                    ;
                    ServiceInterrupts
001D 190B                   btfsc   INTCON,RTIF             ;rtcc interrupt?
001E 2822                   goto    ServiceRTCC             ;yes then service
001F 3020                   movlw   B'00100000'             ;else clr rest
0020 008B                   movwf   INTCON
0021 0009                   retfie
                    ;
                    ServiceRTCC
0022 3060                   movlw   .96                     ;initialize rtcc
0023 0081                   movwf   RTCC
0024 110B                   bcf     INTCON,RTIF             ;clr int flag
0025 2028                   call    IncTimer                ;inc timer
0026 2050                   call    UpdateDisplay           ;update display
0027 0009                   retfie
                    ;
                    ;The display is incremented every 200*5msec = 1 Sec.
                    IncTimer
0028 0A0F                   incf    Count,w                 ;inc count
0029 3AC8                   xorlw   .200                    ;= 200?
002A 1903                   btfsc   STATUS,Z                ;no then skip
002B 282E                   goto    DoIncTime               ;else inc time
002C 0A8F                   incf    Count
002D 0008                   return
                    DoIncTime
002E 018F                   clrf    Count                   ;clr count
002F 0A11                   incf    LsdTime,w               ;get lsd
0030 390F                   andlw   0x0F                    ;mask high nibble
0031 3A0A                   xorlw   0x0a                    ; = 10?
0032 1903                   btfsc   STATUS,Z                ;no then skip
0033 2836                   goto    IncSecondLsd            ;inc next lsd
0034 0A91                   incf    LsdTime                 ;else inc timer
0035 0008                   return
                    IncSecondLsd
0036 0E11                   swapf   LsdTime,w               ;get hi in low nibble
0037 390F                   andlw   0x0F                    ;mask hi nibble
0038 3E01                   addlw   1                       ;inc it
0039 0091                   movwf   LsdTime                 ;restore back
003A 0E91                   swapf   LsdTime                 ;       /
003B 3A0A                   xorlw   0x0a                    ; = 10?
003C 1903                   btfsc   STATUS,Z                ;no then skip
003D 283F                   goto    IncThirdLsd             ;else inc next lsd
003E 0008                   return
                    IncThirdLsd
003F 0191                   clrf    LsdTime
0040 0A10                   incf    MsdTime,w               ;get 3rd lsd
0041 390F                   andlw   0x0F                    ;mask hi nibble
0042 3A0A                   xorlw   0x0a                    ;= 10?
0043 1903                   btfsc   STATUS,Z                ;no then skip
0044 2847                   goto    IncMsd                  ;else Msd
0045 0A90                   incf    MsdTime                 ;else inc timer
0046 0008                   return
                    IncMsd
0047 0E10                   swapf   MsdTime,w               ;get hi in lo nibble
0048 390F                   andlw   0x0F                    ;mask hi nibble
0049 3E01                   addlw   1                       ;inc timer
004A 0090                   movwf   MsdTime                 ;restore back
004B 0E90                   swapf   MsdTime                 ;       /
004C 3A0A                   xorlw   0x0a                    ;= 10?
004D 1903                   btfsc   STATUS,Z                ;no then skip
004E 0190                   clrf    MsdTime                 ;clr msd
004F 0008                   return
```


© 1994 Microchip Technology Inc.

```
                        ;
                        ;
                        UpdateDisplay
0050 0805                       movf    PORT_A,w        ;present sink value in w
0051 0185                       clrf    PORT_A          ;disable all digits sinks
0052 390F                       andlw   0x0f
0053 008C                       movwf   TempC           ;save sink value in tempC
0054 160C                       bsf     TempC,4         ;preset for lsd sink
0055 0C8C                       rrf     TempC           ;determine next sink value
0056 1C03                       btfss   STATUS,CARRY    ;c=1?
0057 118C                       bcf     TempC,3         ;no then reset LSD sink
0058 180C                       btfsc   TempC,0         ;else see if Msd
0059 286B                       goto    UpdateMsd       ;yes then do Msd
005A 188C                       btfsc   TempC,1         ;see if 3rdLsd
005B 2866                       goto    Update3rdLsd    ;yes then do 3rd Lsd
005C 190C                       btfsc   TempC,2         ;see if 2nd Lsd
005D 2861                       goto    Update2ndLsd    ;yes then do 2nd lsd
                        UpdateLsd
005E 0811                       movf    LsdTime,w       ;get Lsd in w
005F 390F                       andlw   0x0f            ;       /
0060 286F                       goto    DisplayOut      ;enable display
                        Update2ndLsd
0061 2080                       call    Chk2LsdZero     ;msd = 0 & 2 lsd 0?
0062 1D03                       btfss   STATUS,Z        ;yes then skip
0063 0E11                       swapf   LsdTime,w       ;get 2nd Lsd in w
0064 390F                       andlw   0x0f            ;mask rest
0065 286F                       goto    DisplayOut      ;enable display
                        Update3rdLsd
0066 2088                       call    ChkMsdZero      ;msd = 0?
0067 1D03                       btfss   STATUS,Z        ;yes then skip
0068 0810                       movf    MsdTime,w       ;get 3rd Lsd in w
0069 390F                       andlw   0x0f            ;mask low nibble
006A 286F                       goto    DisplayOut      ;enable display
                        UpdateMsd
006B 0E10                       swapf   MsdTime,w       ;get Msd in w
006C 390F                       andlw   0x0f            ;mask rest
006D 1903                       btfsc   STATUS,Z        ;msd != 0 then skip
006E 300A                       movlw   0x0a
                        DisplayOut
006F 2074                       call    LedTable        ;get digit output
0070 0086                       movwf   PORT_B          ;drive leds
0071 080C                       movf    TempC,w         ;get sink value in w
0072 0085                       movwf   PORT_A
0073 0008                       return
                        ;
                        ;
                        LedTable
0074 0782                       addwf   PCL             ;add to PC low
0075 343F                       retlw   B'00111111'     ;led drive for 0
0076 3406                       retlw   B'00000110'     ;led drive for 1
0077 345B                       retlw   B'01011011'     ;led drive for 2
0078 344F                       retlw   B'01001111'     ;led drive for 3
0079 3466                       retlw   B'01100110'     ;led drive for 4
007A 346D                       retlw   B'01101101'     ;led drive for 5
007B 347D                       retlw   B'01111101'     ;led drive for 6
007C 3407                       retlw   B'00000111'     ;led drive for 7
007D 347F                       retlw   B'01111111'     ;led drive for 8
007E 3467                       retlw   B'01100111'     ;led drive for 9
007F 3400                       retlw   B'00000000'     ;blank led drive
                        ;
                        ;
                        Chk2LsdZero
0080 2088                       call    ChkMsdZero      ;msd = 0?
0081 1D03                       btfss   STATUS,Z        ;yes then skip
0082 0008                       return                  ;else return
0083 0E11                       swapf   LsdTime,w       ;get 2nd lsd
0084 390F                       andlw   0x0f            ;mask of LSD
0085 1D03                       btfss   STATUS,Z        ;0? then skip
0086 0008                       return
```


© 1994 Microchip Technology Inc.

```
0087 340A                retlw   10                  ;else return with 10
                    ;
                    ChkMsdZero
0088 0810                movf    MsdTime,w           ;get Msd in w
0089 1D03                btfss   STATUS,Z            ;= 0? skip
008A 0008                return                      ;else return
008B 340A                retlw   .10                 ;ret with 10
                    ;
                         end
                    ;
```


© 1994 Microchip Technology Inc.

Appendix B

```
MPASM B0.50                                              PAGE  1

                    ;*****************************************************************
                    ;This program is to demonstrate how to multiplex four 7 segment LED
                    ;digits and a 4X4 keypad using a PIC16C71.
                    ;The four digits will start as '0000' and when a key is hit
                    ;it is displayed on the 7 segment leds as a hex value 0 to F. The last
                    ;digit hit is always displayed on the right most led with the rest of
                    ;the digits shifted to the left. The left most digit is deleted.
                    ;The LEDs are updated every 20msec, the keypad is scanned at a rate of 20
msec.
                    ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                    ;
                    ;                                    Stan D'Souza 5/8/93
                    ;*****************************************************************
                          LIST P=16C71, F=INHX8M
                    ;
                          include "picreg.equ"

                    ;
000C                TempC       equ     0x0c        ;temp general purpose regs
000D                TempD       equ     0x0d
000E                TempE       equ     0x0e
0020                PABuf       equ     0x20
0021                PBBuf       equ     0x21
000F                Count       equ     0x0f        ;count
0010                MsdTime     equ     0x10        ;most significant Timer
0011                LsdTime     equ     0x11        ;Least significant Timer
0012                KeyFlag     equ     0x12        ;flags related to key pad
0000                keyhit      equ     0           ;bit 0 -> key-press on
0001                DebnceOn    equ     1           ;bit 1 -> debounce on
0002                noentry     equ     2           ;no key entry = 0
0003                ServKey     equ     3           ;bit 3 -> service key
0013                Debnce      equ     0x13        ;debounce counter
0014                NewKey      equ     0x14
002F                WBuffer     equ     0x2f
002E                StatBuffer  equ     0x2e
0001                OptionReg   equ     1
0002                PCL         equ     2
                    ;
                    ;
                    push        macro
                                movwf   WBuffer     ;save w reg in Buffer
                                swapf   WBuffer     ;swap it
                                swapf   STATUS,w    ;get status
                                movwf   StatBuffer  ;save it
                                endm
                    ;
                    pop         macro
                                swapf   StatBuffer,w    ;restore status
                                movwf   STATUS          ;
                                swapf   WBuffer,w       ;restore W reg
                                endm
                    ;
                                org     0
0000 280D                       goto    Start       ;skip over interrupt vector
                    ;
                                org     4
                    ;It is always a good practice to save and restore the w reg,
                    ;and the status reg during a interrupt.
                                push
0004 00AF                       movwf   WBuffer     ;save w reg in Buffer
0005 0EAF                       swapf   WBuffer     ;swap it
0006 0E03                       swapf   STATUS,w    ;get status
0007 00AE                       movwf   StatBuffer  ;save it
0008 2036                       call    ServiceInterrupts
```

© 1994 Microchip Technology Inc.

```
                        pop
0009 0E2E               swapf   StatBuffer,w        ;restore status
000A 0083               movwf   STATUS              ;       /
000B 0E2F               swapf   WBuffer,w           ;restore W reg

000C 0009               retfie
                        ;
                        Start
000D 2020               call    InitPorts
000E 202A               call    InitTimers
                        loop
000F 1992               btfsc   KeyFlag,ServKey     ;key service pending
0010 2012               call    ServiceKey          ;yes then service
0011 280F               goto    loop
                        ;
                        ;ServiceKey, does the software service for a keyhit. After a key service,
                        ;the ServKey flag is reset, to denote a completed operation.
                        ServiceKey
0012 0814               movf    NewKey,w            ;get key value
0013 008E               movwf   TempE               ;save in TempE
0014 0E10               swapf   MsdTime,w           ;move MSD out
0015 39F0               andlw   B'11110000'         ;clr lo nibble
0016 0090               movwf   MsdTime             ;save back
0017 0E11               swapf   LsdTime,w           ;get Lsd
0018 390F               andlw   B'00001111'         ;mask off lsd
0019 0490               iorwf   MsdTime             ;and left shift 3rd
001A 0E11               swapf   LsdTime,w           ;get Lsd again
001B 39F0               andlw   B'11110000'         ;mask off 2nd
001C 040E               iorwf   TempE,w             ;or with new lsd
001D 0091               movwf   LsdTime             ;make Lsd
001E 1192               bcf     KeyFlag,ServKey     ;reset service flag
001F 0008               return
                        ;
                        InitPorts
0020 1683               bsf     STATUS,RP0          ;select pg 1
0021 3003               movlw   3                   ;make RA0-3 digital I/O
0022 0108               movwf   ADCON1              ;       /
0023 0205               clrf    TRISA               ;make RA0-4 outputs
0024 0206               clrf    TRISB               ;make RB0-7 outputs
0025 1283               bcf     STATUS,RP0          ;select page 0
0026 0185               clrf    PORT_A              ;make all outputs low
0027 0186               clrf    PORT_B              ;       /
0028 1585               bsf     PORT_A,3            ;enable MSB digit sink
0029 0008               return
                        ;
                        ;
                        ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                        ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                        ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                        ;end result is that we get a rtcc interrupt every 5msec.
                        InitTimers
002A 0190               clrf    MsdTime             ;clr timers
002B 0191               clrf    LsdTime             ;       /
002C 0192               clrf    KeyFlag             ;clr all flags
002D 1683               bsf     STATUS,RP0          ;select pg 1
002E 3084               movlw   B'10000100'         ;assign ps to rtcc
002F 0081               movwf   OptionReg           ;ps = 32
0030 1283               bcf     STATUS,RP0          ;select pg 0
0031 3020               movlw   B'00100000'         ;enable rtcc interrupt
0032 008B               movwf   INTCON              ;
0033 3060               movlw   .96                 ;preload rtcc
0034 0081               movwf   RTCC                ;start counter
0035 0009               retfie
                        ;
                        ServiceInterrupts
0036 190B               btfsc   INTCON,RTIF         ;rtcc interrupt?
0037 283B               goto    ServiceRTCC         ;yes then service
0038 018B               clrf    INTCON              ;else clr all int
0039 168B               bsf     INTCON,RTIE
```


© 1994 Microchip Technology Inc.

```
003A 0008                       return
                        ;
                        ServiceRTCC
003B 3060                       movlw   .96             ;initialize rtcc
003C 0081                       movwf   RTCC
003D 110B                       bcf     INTCON,RTIF     ;clr int flag
003E 1805                       btfsc   PORT_A,0        ;if msb on then do
003F 2042                       call    ScanKeys        ;do a quick key scan
0040 20A1                       call    UpdateDisplay   ;update display
0041 0008                       return
                        ;
                        ;
                        ;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
                        ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                        ;Debounce for a given keyhit is also taken care of.
                        ;The rate of key scan is 20msec with a 4.096Mhz clock.
                        ScanKeys
0042 1C92                       btfss   KeyFlag,DebnceOn        ;debounce on?
0043 2848                       goto    Scan1                   ;no then scan keypad
0044 0B93                       decfsz  Debnce                  ;else dec debounce time
0045 0008                       return                          ;not over then return
0046 1092                       bcf     KeyFlag,DebnceOn        ;over, clr debounce flag
0047 0008                       return                          ;and return
                        Scan1
0048 208A                       call    SavePorts               ;save port values
0049 30EF                       movlw   B'11101111'             ;init TempD
004A 008D                       movwf   TempD
                        ScanNext
004B 0806                       movf    PORT_B,w                ;read to init port
004C 100B                       bcf     INTCON,RBIF             ;clr flag
004D 0C8D                       rrf     TempD                   ;get correct column
004E 1C03                       btfss   STATUS,C                ;if carry set?
004F 2862                       goto    NoKey                   ;no then end
0050 080D                       movf    TempD,w                 ;else output
0051 0086                       movwf   PORT_B                  ;low column scan line
0052 0000                       nop
0053 1C0B                       btfss   INTCON,RBIF             ;flag set?
0054 284B                       goto    ScanNext                ;no then next
0055 1812                       btfsc   KeyFlag,keyhit          ;last key released?
0056 2860                       goto    SKreturn                ;no then exit
0057 1412                       bsf     KeyFlag,keyhit          ;set new key hit
0058 0E06                       swapf   PORT_B,w                ;read port
0059 008E                       movwf   TempE                   ;save in TempE
005A 2064                       call    GetKeyValue             ;get key value 0 - F
005B 0094                       movwf   NewKey                  ;save as New key
005C 1592                       bsf     KeyFlag,ServKey         ;set service flag
005D 1492                       bsf     KeyFlag,DebnceOn        ;set flag
005E 3004                       movlw   4
005F 0093                       movwf   Debnce                  ;load debounce time
                        SKreturn
0060 2097                       call    RestorePorts            ;restore ports
0061 0008                       return
                        ;
                        NoKey
0062 1012                       bcf     KeyFlag,keyhit          ;clr flag
0063 2860                       goto    SKreturn
                        ;
                        ;GetKeyValue gets the key as per the following layout
                        ;
                        ;                Col1    Col2    Col3    Col3
                        ;                (RB3)   (RB2)   (RB1)   (RB0)
                        ;
                        ;Row1(RB4)         0       1       2       3
                        ;
                        ;Row2(RB5)         4       5       6       7
                        ;
                        ;Row3(RB6)         8       9       A       B
                        ;
                        ;Row4(RB7)         C       D       E       F
```

```
                        ;
                        GetKeyValue
0064 018C                       clrf    TempC
0065 1D8D                       btfss   TempD,3         ;first column
0066 286E                       goto    RowValEnd
0067 0A8C                       incf    TempC
0068 1D0D                       btfss   TempD,2         ;second col.
0069 286E                       goto    RowValEnd
006A 0A8C                       incf    TempC
006B 1C8D                       btfss   TempD,1         ;3rd col.
006C 286E                       goto    RowValEnd
006D 0A8C                       incf    TempC           ;last col.
                        RowValEnd
006E 1C0E                       btfss   TempE,0         ;top row?
006F 2878                       goto    GetValCom       ;yes then get 0,1,2&3
0070 1C8E                       btfss   TempE,1         ;2nd row?
0071 2877                       goto    Get4567         ;yes the get 4,5,6&7
0072 1D0E                       btfss   TempE,2         ;3rd row?
0073 2875                       goto    Get89ab         ;yes then get 8,9,a&b
                        Getcdef
0074 150C                       bsf     TempC,2         ;set msb bits
                        Get89ab
0075 158C                       bsf     TempC,3         ;       /
0076 2878                       goto    GetValCom       ;do common part
                        Get4567
0077 150C                       bsf     TempC,2
                        GetValCom
0078 080C                       movf    TempC,w
0079 0782                       addwf   PCL
007A 3400                       retlw   0
007B 3401                       retlw   1
007C 3402                       retlw   2
007D 3403                       retlw   3
007E 3404                       retlw   4
007F 3405                       retlw   5
0080 3406                       retlw   6
0081 3407                       retlw   7
0082 3408                       retlw   8
0083 3409                       retlw   9
0084 340A                       retlw   0a
0085 340B                       retlw   0b
0086 340C                       retlw   0c
0087 340D                       retlw   0d
0088 340E                       retlw   0e
0089 340F                       retlw   0f
                        ;
                        ;SavePorts, saves the porta and portb condition during a key scan
                        ;operation.
                        SavePorts
008A 0805                       movf    PORT_A,w        ;Get sink value
008B 00A0                       movwf   PABuf           ;save in buffer
008C 0185                       clrf    PORT_A          ;disable all sinks
008D 0806                       movf    PORT_B,w        ;get port b
008E 00A1                       movwf   PBBuf           ;save in buffer
008F 30FF                       movlw   0xff            ;make all high
0090 0086                       movwf   PORT_B          ;on port b
0091 1683                       bsf     STATUS,RP0      ;select page 1
0092 1381                       bcf     OptionReg,7     ;enable pull ups
0093 30F0                       movlw   B'11110000'     ;port b hi nibble inputs
0094 0106                       movwf   TRISB           ;lo nibble outputs
0095 1283                       bcf     STATUS,RP0      ;page 0
0096 0008                       return
                        ;
                        ;RestorePorts, restores the condition of porta and portb after a
                        ;key scan operation.
                        RestorePorts
0097 0821                       movf    PBBuf,w         ;get port n
0098 0086                       movwf   PORT_B
0099 0820                       movf    PABuf,w         ;get port a value
```


© 1994 Microchip Technology Inc.

```
009A 0085                       movwf   PORT_A
009B 1683                       bsf     STATUS,RP0      ;select page 1
009C 1781                       bsf     OptionReg,7     ;disable pull ups
009D 0205                       clrf    TRISA           ;make port a outputs
009E 0206                       clrf    TRISB           ;as well as PORTB
009F 1283                       bcf     STATUS,RP0      ;page 0
00A0 0008                       return
                        ;
                        ;
                        UpdateDisplay
00A1 0805                       movf    PORT_A,w        ;present sink value in w
00A2 0185                       clrf    PORT_A          ;disable all digits sinks
00A3 390F                       andlw   0x0f
00A4 008C                       movwf   TempC           ;save sink value in tempC
00A5 160C                       bsf     TempC,4         ;preset for lsd sink
00A6 0C8C                       rrf     TempC           ;determine next sink value
00A7 1C03                       btfss   STATUS,CARRY    ;c=1?
00A8 118C                       bcf     TempC,3         ;no then reset LSD sink
00A9 180C                       btfsc   TempC,0         ;else see if Msd
00AA 28B8                       goto    UpdateMsd       ;yes then do Msd
00AB 188C                       btfsc   TempC,1         ;see if 3rdLsd
00AC 28B5                       goto    Update3rdLsd    ;yes then do 3rd Lsd
00AD 190C                       btfsc   TempC,2         ;see if 2nd Lsd
00AE 28B2                       goto    Update2ndLsd    ;yes then do 2nd lsd
                        UpdateLsd
00AF 0811                       movf    LsdTime,w       ;get Lsd in w
00B0 390F                       andlw   0x0f            ;       /
00B1 28BA                       goto    DisplayOut
                        Update2ndLsd
00B2 0E11                       swapf   LsdTime,w       ;get 2nd Lsd in w
00B3 390F                       andlw   0x0f            ;mask rest
00B4 28BA                       goto    DisplayOut      ;enable display
                        Update3rdLsd
00B5 0810                       movf    MsdTime,w       ;get 3rd Lsd in w
00B6 390F                       andlw   0x0f            ;mask low nibble
00B7 28BA                       goto    DisplayOut      ;enable display
                        UpdateMsd
00B8 0E10                       swapf   MsdTime,w       ;get Msd in w
00B9 390F                       andlw   0x0f            ;mask rest
                        DisplayOut
00BA 20BF                       call    LedTable        ;get digit output
00BB 0086                       movwf   PORT_B          ;drive leds
00BC 080C                       movf    TempC,w         ;get sink value in w
00BD 0085                       movwf   PORT_A
00BE 0008                       return
                        ;
                        ;
                        LedTable
00BF 0782                       addwf   PCL             ;add to PC low
00C0 343F                       retlw   B'00111111'     ;led drive for 0
00C1 3406                       retlw   B'00000110'     ;led drive for 1
00C2 345B                       retlw   B'01011011'     ;led drive for 2
00C3 344F                       retlw   B'01001111'     ;led drive for 3
00C4 3466                       retlw   B'01100110'     ;led drive for 4
00C5 346D                       retlw   B'01101101'     ;led drive for 5
00C6 347D                       retlw   B'01111101'     ;led drive for 6
00C7 3407                       retlw   B'00000111'     ;led drive for 7
00C8 347F                       retlw   B'01111111'     ;led drive for 8
00C9 3467                       retlw   B'01100111'     ;led drive for 9
00CA 3477                       retlw   B'01110111'     ;led drive for A
00CB 347C                       retlw   B'01111100'     ;led drive for b
00CC 3439                       retlw   B'00111001'     ;led drive for C
00CD 345E                       retlw   B'01011110'     ;led drive for d
00CE 3479                       retlw   B'01111001'     ;led drive for E
00CF 3471                       retlw   B'01110001'     ;led drive for F
                        ;
                        ;
                                end
                        ;
```


© 1994 Microchip Technology Inc.

Appendix C

```
MPASM B0.50                                              PAGE  1


                        ;*****************************************************************
                        ;This program is to demonstrate how to multiplex four 7 segment LED
                        ;and sample ch0 of the a/d in a PIC16C71. The a/d value is displayed
                        ;as a 3 digit decimal value of the a/d input (0 - 255).
                        ;The LEDs are updated every 20msec, the a/d is sampled every 20 msec.
                        ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                        ;
                        ;                                        Stan D'Souza 5/8/93
                        ;*****************************************************************
                              LIST P=16C71, F=INHX8M
                        ;
                              include "picreg.equ"

                        ;
0026                    BcdMsd          equ     26
0027                    Bcd             equ     27
000C                    TempC           equ     0x0c            ;temp general purpose regs
000D                    TempD           equ     0x0d
000E                    TempE           equ     0x0e
0020                    PABuf           equ     0x20
0021                    PBBuf           equ     0x21
000F                    Count           equ     0x0f            ;count
0010                    MsdTime         equ     0x10            ;most significant Timer
0011                    LsdTime         equ     0x11            ;Least significant Timer
0012                    ADFlag          equ     0x12            ;flags related to key pad
0005                    ADOver          equ     5               ;bit 5 -> a/d over
002F                    WBuffer         equ     0x2f
002E                    StatBuffer      equ     0x2e
0001                    OptionReg       equ     1
0002                    PCL             equ     2
                        ;
                        push            macro
                        movwf           WBuffer                 ;save w reg in Buffer
                        swapf           WBuffer                 ;swap it
                        swapf           STATUS,w                ;get status
                        movwf           StatBuffer              ;save it
                        endm
                        ;
                        pop             macro
                        swapf           StatBuffer,w            ;restore status
                        movwf           STATUS                  ;       /
                        swapf           WBuffer,w               ;restore W reg
                        endm
                        ;
                        org             0
0000 280D               goto            Start                   ;skip over interrupt vector
                        ;
                        org             4
                        ;It is always a good practice to save and restore the w reg,
                        ;and the status reg during a interrupt.
                        push
0004 00AF               movwf           WBuffer                 ;save w reg in Buffer
0005 0EAF               swapf           WBuffer                 ;swap it
0006 0E03               swapf           STATUS,w                ;get status
0007 00AE               movwf           StatBuffer              ;save it

0008 2039               call            ServiceInterrupts
                        pop
0009 0E2E               swapf           StatBuffer,w            ;restore status
000A 0083                     movwf     STATUS          ;       /
```


© 1994 Microchip Technology Inc.

```
000B 0E2F                       swapf   WBuffer,w               ;restore W reg

000C 0009                       retfie
                        ;
                        Start
000D 2021                       call    InitPorts
000E 202B                       call    InitTimers
000F 2036                       call    InitAd
                        loop
0010 1A92                       btfsc   ADFlag,ADOver           ;a/d over?
0011 2013                       call    UpdateAd                ;yes then update
0012 2810                       goto    loop
                        ;
                        UpdateAd
0013 1C88                       btfss   ADCON0,ADIF             ;a/d done?
0014 0008                       return                          ;no then leave
0015 0809                       movf    ADRES,W                 ;get a/d value
0016 00A1                       movwf   L_byte
0017 01A0                       clrf    H_byte
0018 20AD                       call    B2_BCD
0019 0824                       movf    R2,W                    ;get LSd
001A 0091                       movwf   LsdTime                 ;save in LSD
001B 0823                       movf    R1,W                    ;get Msd
001C 0090                       movwf   MsdTime                 ;save in Msd
001D 1088                       bcf     ADCON0,ADIF             ;clr interrupt flag
001E 1008                       bcf     ADCON0,ADON             ;turn off a/d
001F 1292                       bcf     ADFlag,ADOver           ;clr flag
0020 0008                       return
                        ;
                        ;
                        ;
                        InitPorts
0021 1683                       bsf     STATUS,RP0              ;select pg 1
0022 3003                       movlw   3                       ;make RA0-3 digital I/O
0023 0108                       movwf   ADCON1                  ;       /
0024 0205                       clrf    TRISA                   ;make RA0-4 outputs
0025 0206                       clrf    TRISB                   ;make RB0-7 outputs
0026 1283                       bcf     STATUS,RP0              ;select page 0
0027 0185                       clrf    PORT_A                  ;make all outputs low
0028 0186                       clrf    PORT_B                  ;       /
0029 1585                       bsf     PORT_A,3                ;enable MSB digit sink
002A 0008                       return
                        ;
                        ;
                        ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                        ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                        ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                        ;end result is that we get a rtcc interrupt every 5msec.
                        InitTimers
002B 0190                       clrf    MsdTime                 ;clr timers
002C 0191                       clrf    LsdTime                 ;       /
002D 1683                       bsf     STATUS,RP0              ;select pg 1
002E 3084                       movlw   B'10000100'             ;assign ps to rtcc
002F 0081                       movwf   OptionReg               ;ps = 32
0030 1283                       bcf     STATUS,RP0              ;select pg 0
0031 3020                       movlw   B'00100000'             ;enable rtcc interrupt
0032 008B                       movwf   INTCON                  ;
0033 3060                       movlw   .96                     ;preload rtcc
0034 0081                       movwf   RTCC                    ;start counter
0035 0009                       retfie
                        ;
                        ;
                        InitAd
0036 30C8                       movlw   B'11001000'             ;init a/d
0037 0088                       movwf   ADCON0
0038 0008                       return
                        ;
```

© 1994 Microchip Technology Inc.

```
                        ;
                        ServiceInterrupts
0039 190B                       btfsc   INTCON,RTIF            ;rtcc interrupt?
003A 283E                       goto    ServiceRTCC            ;yes then service
003B 018B                       clrf    INTCON
003C 168B                       bsf     INTCON,RTIE
003D 0008                       return
                        ;
                        ServiceRTCC
003E 3060                       movlw   .96                    ;initialize rtcc
003F 0081                       movwf   RTCC
0040 110B                       bcf     INTCON,RTIF            ;clr int flag
0041 1C05                       btfss   PORT_A,0               ;last digit?
0042 2045                       call    SampleAd               ;then sample a/d
0043 2071                       call    UpdateDisplay          ;else update display
0044 0008                       return
                        ;
                        ;
                        SampleAd
0045 205A                       call    SavePorts
0046 204C                       call    DoAd                   ;do a ad conversion
                        AdDone
0047 1908                       btfsc   ADCON0,GO              ;ad done?
0048 2847                       goto    AdDone                 ;no then loop
0049 1692                       bsf     ADFlag,ADOver          ;set a/d over flag
004A 2067                       call    RestorePorts           ;restore ports
004B 0008                       return
                        ;
                        ;
                        DoAd
004C 0186                       clrf    PORT_B                 ;turn off leds
004D 1683                       bsf     STATUS,RP0             ;select pg 1
004E 300F                       movlw   0x0f                   ;make port a hi-Z
004F 0105                       movwf   TRISA                  ;       /
0050 1283                       bcf     STATUS,RP0             ;select pg 0
0051 1408                       bsf     ADCON0,ADON            ;start a/d
0052 307D                       movlw   .125
0053 2056                       call    Wait
0054 1508                       bsf     ADCON0,GO              ;start conversion
0055 0008                       return
                        ;
                        ;
                        Wait
0056 008C                       movwf   TempC                  ;store in temp
                        Next
0057 0B8C                       decfsz  TempC
0058 2857                       goto    Next
0059 0008                       return

                        ;
                        ;SavePorts, saves the porta and portb condition during a key scan
                        ;operation.
                        SavePorts
005A 0805                       movf    PORT_A,w               ;Get sink value
005B 00A0                       movwf   PABuf                  ;save in buffer
005C 0185                       clrf    PORT_A                 ;disable all sinks
005D 0806                       movf    PORT_B,w               ;get port b
005E 00A1                       movwf   PBBuf                  ;save in buffer
005F 30FF                       movlw   0xff                   ;make all high
0060 0086                       movwf   PORT_B                 ;on port b
0061 1683                       bsf     STATUS,RP0             ;select page 1
0062 1381                       bcf     OptionReg,7            ;enable pull ups
0063 30F0                       movlw   B'11110000'            ;port b hi nibble inputs
0064 0106                       movwf   TRISB                  ;lo nibble outputs
0065 1283                       bcf     STATUS,RP0             ;page 0
0066 0008                       return
                        ;
                        ;RestorePorts, restores the condition of porta and portb after a
                        ;key scan operation.
```


© 1994 Microchip Technology Inc.

```
                        RestorePorts
0067 0821                       movf    PBBuf,w         ;get port n
0068 0086                       movwf   PORT_B
0069 0820                       movf    PABuf,w         ;get port a value
006A 0085                       movwf   PORT_A
006B 1683                       bsf     STATUS,RP0      ;select page 1
006C 1781                       bsf     OptionReg,7     ;disable pull ups
006D 0205                       clrf    TRISA           ;make port a outputs
006E 0206                       clrf    TRISB           ;as well as PORTB
006F 1283                       bcf     STATUS,RP0      ;page 0
0070 0008                       return
                        ;
                        ;
                        UpdateDisplay
0071 0805                       movf    PORT_A,w        ;present sink value in w
0072 0185                       clrf    PORT_A          ;disable all digits sinks
0073 390F                       andlw   0x0f
0074 008C                       movwf   TempC           ;save sink value in tempC
0075 160C                       bsf     TempC,4         ;preset for lsd sink
0076 0C8C                       rrf     TempC           ;determine next sink value
0077 1C03                       btfss   STATUS,CARRY    ;c=1?
0078 118C                       bcf     TempC,3         ;no then reset LSD sink
0079 180C                       btfsc   TempC,0         ;else see if Msd
007A 288C                       goto    UpdateMsd       ;yes then do Msd
007B 188C                       btfsc   TempC,1         ;see if 3rdLsd
007C 2887                       goto    Update3rdLsd    ;yes then do 3rd Lsd
007D 190C                       btfsc   TempC,2         ;see if 2nd Lsd
007E 2882                       goto    Update2ndLsd    ;yes then do 2nd lsd
                        UpdateLsd
007F 0811                       movf    LsdTime,w       ;get Lsd in w
0080 390F                       andlw   0x0f            ;       /
0081 2890                       goto    DisplayOut      ;enable display
                        Update2ndLsd
0082 20A1                       call    Chk2LsdZero     ;msd = 0 & 2 lsd 0?
0083 1D03                       btfss   STATUS,Z        ;yes then skip
0084 0E11                       swapf   LsdTime,w       ;get 2nd Lsd in w
0085 390F                       andlw   0x0f            ;mask rest
0086 2890                       goto    DisplayOut      ;enable display
                        Update3rdLsd
0087 20A9                       call    ChkMsdZero      ;msd = 0?
0088 1D03                       btfss   STATUS,Z        ;yes then skip
0089 0810                       movf    MsdTime,w       ;get 3rd Lsd in w
008A 390F                       andlw   0x0f            ;mask low nibble
008B 2890                       goto    DisplayOut      ;enable display
                        UpdateMsd
008C 0E10                       swapf   MsdTime,w       ;get Msd in w
008D 390F                       andlw   0x0f            ;mask rest
008E 1903                       btfsc   STATUS,Z        ;msd != 0 then skip
008F 300A                       movlw   0x0a
                        DisplayOut
0090 2095                       call    LedTable        ;get digit output
0091 0086                       movwf   PORT_B          ;drive leds
0092 080C                       movf    TempC,w         ;get sink value in w
0093 0085                       movwf   PORT_A
0094 0008                       return
                        ;
                        ;
                        LedTable
0095 0782                       addwf   PCL             ;add to PC low
0096 343F                       retlw   B'00111111'     ;led drive for 0
0097 3406                       retlw   B'00000110'     ;led drive for 1
0098 345B                       retlw   B'01011011'     ;led drive for 2
0099 344F                       retlw   B'01001111'     ;led drive for 3
009A 3466                       retlw   B'01100110'     ;led drive for 4
009B 346D                       retlw   B'01101101'     ;led drive for 5
009C 347D                       retlw   B'01111101'     ;led drive for 6
009D 3407                       retlw   B'00000111'     ;led drive for 7
009E 347F                       retlw   B'01111111'     ;led drive for 8
009F 3467                       retlw   B'01100111'     ;led drive for 9
```


© 1994 Microchip Technology Inc.

```
00A0 3400                retlw          B'00000000'    ;blank led drive
                    ;
                    ;
                    Chk2LsdZero
00A1 20A9                call           ChkMsdZero     ;msd = 0?
00A2 1D03                btfss          STATUS,Z       ;yes then skip
00A3 0008                return                        ;else return
00A4 0E11                swapf          LsdTime,w      ;get 2nd lsd
00A5 390F                andlw          0x0f           ;mask of LSD
00A6 1D03                btfss          STATUS,Z       ;0? then skip
00A7 0008                return
00A8 340A                retlw          .10            ;else return with 10
                    ;
                    ChkMsdZero
00A9 0810                movf           MsdTime,w      ;get Msd in w
00AA 1D03                btfss          STATUS,Z       ;= 0? skip
00AB 0008                return                        ;else return
00AC 340A                retlw          .10            ;ret with 10
                    ;
                    ;
                    ;
0026                count   equ     26
0027                temp    equ     27
                    ;
0020                H_byte  equ     20
0021                L_byte  equ     21
0022                R0      equ     22                ; RAM Assignments
0023                R1      equ     23
0024                R2      equ     24
                    ;
                    ;
00AD 1003           B2_BCD  bcf     STATUS,0          ; clear the carry bit
00AE 3010                   movlw   .16
00AF 00A6                   movwf   count
00B0 01A2                   clrf    R0
00B1 01A3                   clrf    R1
00B2 01A4                   clrf    R2
00B3 0DA1           loop16  rlf     L_byte
00B4 0DA0                   rlf     H_byte
00B5 0DA4                   rlf     R2
00B6 0DA3                   rlf     R1
00B7 0DA2                   rlf     R0
                    ;
00B8 0BA6                   decfsz  count
00B9 28BB                   goto    adjDEC
00BA 3400                   RETLW   0
                    ;
00BB 3024           adjDEC  movlw   R2
00BC 0084                   movwf   FSR
00BD 20C5                   call    adjBCD
                    ;
00BE 3023                   movlw   R1
00BF 0084                   movwf   FSR
00C0 20C5                   call    adjBCD
                    ;
00C1 3022                   movlw   R0
00C2 0084                   movwf   FSR
00C3 20C5                   call    adjBCD
                    ;
00C4 28B3                   goto    loop16
                    ;
00C5 3003           adjBCD  movlw   3
00C6 0700                   addwf   0,W
00C7 00A7                   movwf   temp
00C8 19A7                   btfsc   temp,3            ; test if result > 7
00C9 0080                   movwf   0
00CA 3030                   movlw   30
00CB 0700                   addwf   0,W
00CC 00A7                   movwf   temp
```


© 1994 Microchip Technology Inc.

```
00CD 1BA7                btfsc   temp,7          ; test if result > 7
00CE 0080                movwf   0               ; save as MSD
00CF 3400                RETLW   0
                         ;
                         ;

                         end

                         ;
```

APPENDIX D

```
MPASM 1.00 Released    MPLXAD.ASM   7-15-1994  13:43:14                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;*************************************************************************
                     0002 ;This program is to demonstrate how to multiplex four 7 segment LED
                     0003 ;digits and a 4X4 keypad along with 4 A/D inputs using a PIC16C71.
                     0004 ;The four digits will first display the decimal a/d value of ch0.
                     0005 ;When keys from 0 - 3 are hit the corresponding channel's a/d value
                     0006 ;is displayed in decimal.
                     0007 ;The LEDs are updated every 20mS, the keypad is scanned at a rate of 20 mS.
                     0008 ;All 4 channels are scanned at 20mS rate, so each channel gets scanned
                     0009 ;every 80mS. A faster rate of scanning is possible as required by
                     0010 ;the users application.
                     0011 ;The RTCC timer is used in internal interrupt mode to generate the
                     0012 ;5 mS.
                     0013 ;
                     0014 ;                                        Stan D'Souza 5/8/93
                     0015 ;
                     0016 ;Corrected error in display routine.
                     0017 ;                                        Stan D'Souza 2/27/94
                     0018 ;*************************************************************************
                     0019         LIST P=16C71, F=INHX8M
                     0020 ;
                     0021         include "picreg.equ"
                     0083
                     0084
                     0021
                     0022 ;
000C                 0023 TempC    equ     0x0c            ;temp general purpose regs
000D                 0024 TempD    equ     0x0d
000E                 0025 TempE    equ     0x0e
0020                 0026 PABuf    equ     0x20
0021                 0027 PBBuf    equ     0x21
000F                 0028 Count    equ     0x0f           ;count
0010                 0029 MsdTime equ      0x10            ;most significant Timer
0011                 0030 LsdTime equ      0x11            ;Least significant Timer
                     0031 ;
0012                 0032 Flag     equ     0x12            ;general purpose flag reg
0001                 0033  #define keyhit  Flag,0         ;bit 0 -> key-press on
0002                 0034  #define DebnceOn Flag,1        ;bit 1 -> debounce on
0003                 0035  #define noentry Flag,2         ;no key entry = 0
0004                 0036  #define ServKey Flag,3         ;bit 3 -> service key
0005                 0037  #define ADOver  Flag,4         ;bit 4 -> a/d conv. over
                     0038 ;
0013                 0039 Debnce  equ      0x13            ;debounce counter
0014                 0040 NewKey  equ      0x14
0015                 0041 DisplayCh equ    0x15            ;channel to be displayed
                     0042 ;
0016                 0043 ADTABLE equ      0x16            ;4 locations are reserved here
                     0044                                  ;from 0x16 to 0x19
                     0045 ;
002F                 0046 WBuffer equ      0x2f
002E                 0047 StatBuffer equ   0x2e
0001                 0048 OptionReg equ    1
0002                 0049 PCL      equ     2
                     0050 ;
                     0051 ;
                     0052 push     macro
                     0053         movwf   WBuffer         ;save w reg in Buffer
                     0054         swapf   WBuffer         ;swap it
                     0055         swapf   STATUS,w        ;get status
                     0056         movwf   StatBuffer      ;save it
                     0057         endm
                     0058 ;
```


© 1994 Microchip Technology Inc.

```
                    0059 pop    macro
                    0060        swapf   StatBuffer,w    ;restore status
                    0061        movwf   STATUS          ;    /
                    0062        swapf   WBuffer,w       ;restore W reg
                    0063        endm
                    0064 ;
                    0065        org     0
0000 280D           0066        goto    Start           ;skip over interrupt vector
                    0067 ;
                    0068        org     4
                    0069 ;It is always a good practice to save and restore the w reg,
                    0070 ;and the status reg during a interrupt.
                    0071        push
0004 00AF             M         movwf   WBuffer         ;save w reg in Buffer
0005 0EAF             M         swapf   WBuffer         ;swap it
0006 0E03             M         swapf   STATUS,w        ;get status
0007 00AE             M         movwf   StatBuffer      ;save it
0008 2052           0072        call    ServiceInterrupts
                    0073        pop
0009 0E2E             M         swapf   StatBuffer,w    ;restore status
000A 0083             M         movwf   STATUS          ;    /
000B 0E2F             M         swapf   WBuffer,w       ;restore W reg
000C 0009           0074        retfie
                    0075 ;
                    0076 Start
000D 203B           0077        call    InitPorts
000E 20EE           0078        call    InitAd
000F 2045           0079        call    InitTimers
                    0080 loop
0010 1992           0081        btfsc   ServKey         ;key service pending
0011 2015           0082        call    ServiceKey      ;yes then service
0012 1A12           0083        btfsc   ADOver          ;a/d pending?
0013 2028           0084        call    ServiceAD       ;yes the service a/d
0014 2810           0085        goto    loop
                    0086 ;
                    0087 ;ServiceKey, does the software service for a keyhit. After a key service,
                    0088 ;the ServKey flag is reset, to denote a completed operation.
                    0089 ServiceKey
0015 1192           0090        bcf     ServKey         ;reset service flag
0016 0814           0091        movf    NewKey,w        ;get key value
0017 3C03           0092        sublw   3               ;key > 3?
0018 1C03           0093        btfss   STATUS,C        ;no then skip
0019 0008           0094        return                  ;else ignore key
001A 0814           0095        movf    NewKey,w
001B 0095           0096        movwf   DisplayCh       ;load new channel
                    0097 ;
                    0098 LoadAD
001C 3016           0099        movlw   ADTABLE         ;get top of table
001D 0715           0100        addwf   DisplayCh,w     ;add offset
001E 0084           0101        movwf   FSR             ;init FSR
001F 0800           0102        movf    0,w             ;get a/d value
0020 00A1           0103        movwf   L_byte
0021 01A0           0104        clrf    H_byte
0022 2106           0105        call    B2_BCD
0023 0824           0106        movf    R2,W            ;get LSd
0024 0091           0107        movwf   LsdTime         ;save in LSD
0025 0823           0108        movf    R1,W            ;get Msd
0026 0090           0109        movwf   MsdTime         ;save in Msd
0027 0008           0110        return
                    0111 ;
                    0112 ;This rountine essentially loads the ADRES value in the table location
                    0113 ;determined by the channel offset. If channel 0 then ADRES is saved
                    0114 ;in location ADTABLE. If channel 1 then ADRES is saved at ADTABLE + 1.
                    0115 ;and so on.
                    0116 ServiceAD
0028 0808           0117        movf    ADCON0,w        ;get adcon0
0029 008C           0118        movwf   TempC           ;save in temp
002A 3008           0119        movlw   B'00001000'     ;select next channel
002B 0708           0120        addwf   ADCON0,w        ;    /
```


© 1994 Microchip Technology Inc.

```
002C 1A88           0121        btfsc   ADCON0,5        ;if <= ch3
002D 30C1           0122        movlw   B'11000001'     ;select ch0
002E 0088           0123        movwf   ADCON0
                    0124        ;now load adres in the table
002F 3016           0125        movlw   ADTABLE
0030 0084           0126        movwf   FSR             ;load FSR with top
0031 0C8C           0127        rrf     TempC
0032 0C8C           0128        rrf     TempC
0033 0C0C           0129        rrf     TempC,w         ;get in w reg
0034 3903           0130        andlw   3               ;mask off all but last 2
0035 0784           0131        addwf   FSR             ;add offset to table
0036 0809           0132        movf    ADRES,w         ;get a/d value
0037 0080           0133        movwf   0               ;load indirectly
0038 1212           0134        bcf     ADOver          ;clear flag
0039 201C           0135        call    LoadAD          ;load a/d value in display reg.
003A 0008           0136        return
                    0137
                    0138
                    0139
                    0140 ;
                    0141 InitPorts
003B 1683           0142        bsf     STATUS,RP0      ;select pg 1
003C 3003           0143        movlw   3               ;make RA0-3 digital I/O
003D 0088           0144        movwf   ADCON1          ;       /
003E 0185           0145        clrf    TRISA           ;make RA0-4 outputs
003F 0186           0146        clrf    TRISB           ;make RB0-7 outputs
0040 1283           0147        bcf     STATUS,RP0      ;select page 0
0041 0185           0148        clrf    PORT_A          ;make all outputs low
0042 0186           0149        clrf    PORT_B          ;       /
0043 1585           0150        bsf     PORT_A,3        ;enable MSB digit sink
0044 0008           0151        return
                    0152 ;
                    0153 ;
                    0154 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                    0155 ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                    0156 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                    0157 ;end result is that we get a rtcc interrupt every 5mS.
                    0158 InitTimers
0045 0190           0159        clrf    MsdTime         ;clr timers
0046 0191           0160        clrf    LsdTime         ;       /
0047 0195           0161        clrf    DisplayCh       ;show channel 0
0048 0192           0162        clrf    Flag            ;clr all flags
0049 1683           0163        bsf     STATUS,RP0      ;select pg 1
004A 3084           0164        movlw   B'10000100'     ;assign ps to rtcc
004B 0081           0165        movwf   OptionReg       ;ps = 32
004C 1283           0166        bcf     STATUS,RP0      ;select pg 0
004D 3020           0167        movlw   B'00100000'     ;enable rtcc interrupt
004E 008B           0168        movwf   INTCON          ;
004F 3060           0169        movlw   .96             ;preload rtcc
0050 0081           0170        movwf   RTCC            ;start counter
0051 0009           0171        retfie
                    0172 ;
                    0173 ServiceInterrupts
0052 190B           0174        btfsc   INTCON,RTIF     ;rtcc interrupt?
0053 2857           0175        goto    ServiceRTCC     ;yes then service
0054 018B           0176        clrf    INTCON          ;else clr all int
0055 168B           0177        bsf     INTCON,RTIE
0056 0008           0178        return
                    0179 ;
                    0180 ServiceRTCC
0057 3060           0181        movlw   .96             ;initialize rtcc
0058 0081           0182        movwf   RTCC
0059 110B           0183        bcf     INTCON,RTIF     ;clr int flag
005A 1805           0184        btfsc   PORT_A,0        ;scan keys every 20 mS
005B 2060           0185        call    ScanKeys        ;when digit 1 is on
005C 1985           0186        btfsc   PORT_A,3        ;scan a/d every 20mS
005D 20F1           0187        call    SampleAd        ;when digit 4 is on
005E 20BF           0188        call    UpdateDisplay   ;update display
005F 0008           0189        return
```


© 1994 Microchip Technology Inc.

```
                        0190 ;
                        0191 ;
                        0192 ;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
                        0193 ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                        0194 ;Debounce for a given keyhit is also taken care of.
                        0195 ;The rate of key scan is 20mS with a 4.096Mhz clock.
                        0196 ScanKeys
0060 1C92               0197         btfss   DebnceOn        ;debounce on?
0061 2866               0198         goto    Scan1           ;no then scan keypad
0062 0B93               0199         decfsz  Debnce          ;else dec debounce time
0063 0008               0200         return                  ;not over then return
0064 1092               0201         bcf     DebnceOn        ;over, clr debounce flag
0065 0008               0202         return                  ;and return
                        0203 Scan1
0066 20A8               0204         call    SavePorts       ;save port values
0067 30EF               0205         movlw   B'11101111'     ;init TempD
0068 008D               0206         movwf   TempD
                        0207 ScanNext
0069 0806               0208         movf    PORT_B,w        ;read to init port
006A 100B               0209         bcf     INTCON,RBIF     ;clr flag
006B 0C8D               0210         rrf     TempD           ;get correct column
006C 1C03               0211         btfss   STATUS,C        ;if carry set?
006D 2880               0212         goto    NoKey           ;no then end
006E 080D               0213         movf    TempD,w         ;else output
006F 0086               0214         movwf   PORT_B          ;low column scan line
0070 0000               0215         nop
0071 1C0B               0216         btfss   INTCON,RBIF     ;flag set?
0072 2869               0217         goto    ScanNext        ;no then next
0073 1812               0218         btfsc   keyhit          ;last key released?
0074 287E               0219         goto    SKreturn        ;no then exit
0075 1412               0220         bsf     keyhit          ;set new key hit
0076 0E06               0221         swapf   PORT_B,w        ;read port
0077 008E               0222         movwf   TempE           ;save in TempE
0078 2082               0223         call    GetKeyValue     ;get key value 0 - F
0079 0094               0224         movwf   NewKey          ;save as New key
007A 1592               0225         bsf     ServKey         ;set service flag
007B 1492               0226         bsf     DebnceOn        ;set flag
007C 3004               0227         movlw   4
007D 0093               0228         movwf   Debnce          ;load debounce time
                        0229 SKreturn
007E 20B5               0230         call    RestorePorts    ;restore ports
007F 0008               0231         return
                        0232 ;
                        0233 NoKey
0080 1012               0234         bcf     keyhit          ;clr flag
0081 287E               0235         goto    SKreturn
                        0236 ;
                        0237 ;GetKeyValue gets the key as per the following layout
                        0238 ;
                        0239 ;                  Col1    Col2    Col3    Col3
                        0240 ;                  (RB3)   (RB2)   (RB1)   (RB0)
                        0241 ;
                        0242 ;Row1(RB4)           0       1       2       3
                        0243 ;
                        0244 ;Row2(RB5)           4       5       6       7
                        0245 ;
                        0246 ;Row3(RB6)           8       9       A       B
                        0247 ;
                        0248 ;Row4(RB7)           C       D       E       F
                        0249 ;
                        0250 GetKeyValue
0082 018C               0251         clrf    TempC
0083 1D8D               0252         btfss   TempD,3         ;first column
0084 288C               0253         goto    RowValEnd
0085 0A8C               0254         incf    TempC
0086 1D0D               0255         btfss   TempD,2         ;second col.
0087 288C               0256         goto    RowValEnd
0088 0A8C               0257         incf    TempC
```

```
0089 1C8D           0258          btfss   TempD,1         ;3rd col.
008A 288C           0259          goto    RowValEnd
008B 0A8C           0260          incf    TempC           ;last col.
                    0261 RowValEnd
008C 1C0E           0262          btfss   TempE,0         ;top row?
008D 2896           0263          goto    GetValCom       ;yes then get 0,1,2&3
008E 1C8E           0264          btfss   TempE,1         ;2nd row?
008F 2895           0265          goto    Get4567         ;yes the get 4,5,6&7
0090 1D0E           0266          btfss   TempE,2         ;3rd row?
0091 2893           0267          goto    Get89ab         ;yes then get 8,9,a&b
                    0268 Getcdef
0092 150C           0269          bsf     TempC,2         ;set msb bits
                    0270 Get89ab
0093 158C           0271          bsf     TempC,3         ;       /
0094 2896           0272          goto    GetValCom       ;do common part
                    0273 Get4567
0095 150C           0274          bsf     TempC,2
                    0275 GetValCom
0096 080C           0276          movf    TempC,w
0097 0782           0277          addwf   PCL
0098 3400           0278          retlw   0
0099 3401           0279          retlw   1
009A 3402           0280          retlw   2
009B 3403           0281          retlw   3
009C 3404           0282          retlw   4
009D 3405           0283          retlw   5
009E 3406           0284          retlw   6
009F 3407           0285          retlw   7
00A0 3408           0286          retlw   8
00A1 3409           0287          retlw   9
00A2 340A           0288          retlw   0a
00A3 340B           0289          retlw   0b
00A4 340C           0290          retlw   0c
00A5 340D           0291          retlw   0d
00A6 340E           0292          retlw   0e
00A7 340F           0293          retlw   0f
                    0294 ;
                    0295 ;SavePorts, saves the porta and portb condition during a key scan
                    0296 ;operation.
                    0297 SavePorts
00A8 0805           0298          movf    PORT_A,w        ;Get sink value
00A9 00A0           0299          movwf   PABuf           ;save in buffer
00AA 0185           0300          clrf    PORT_A          ;disable all sinks
00AB 0806           0301          movf    PORT_B,w        ;get port b
00AC 00A1           0302          movwf   PBBuf           ;save in buffer
00AD 30FF           0303          movlw   0xff            ;make all high
00AE 0086           0304          movwf   PORT_B          ;on port b
00AF 1683           0305          bsf     STATUS,RP0      ;select page 1
00B0 1381           0306          bcf     OptionReg,7     ;enable pull ups
00B1 30F0           0307          movlw   B'11110000'     ;port b hi nibble inputs
00B2 0086           0308          movwf   TRISB           ;lo nibble outputs
00B3 1283           0309          bcf     STATUS,RP0      ;page 0
00B4 0008           0310          return
                    0311 ;
                    0312 ;RestorePorts, restores the condition of porta and portb after a
                    0313 ;key scan operation.
                    0314 RestorePorts
00B5 0821           0315          movf    PBBuf,w         ;get port b
00B6 0086           0316          movwf   PORT_B
00B7 0820           0317          movf    PABuf,w         ;get port a value
00B8 0085           0318          movwf   PORT_A
00B9 1683           0319          bsf     STATUS,RP0      ;select page 1
00BA 1781           0320          bsf     OptionReg,7     ;disable pull ups
00BB 0185           0321          clrf    TRISA           ;make port a outputs
00BC 0186           0322          clrf    TRISB           ;as well as PORTB
00BD 1283           0323          bcf     STATUS,RP0      ;page 0
00BE 0008           0324          return
                    0325 ;
                    0326 ;
```


© 1994 Microchip Technology Inc.

```
                     0327 UpdateDisplay
00BF 0805            0328         movf    PORT_A,w        ;present sink value in w
00C0 0185            0329         clrf    PORT_A          ;disable all digits sinks
00C1 390F            0330         andlw   0x0f
00C2 008C            0331         movwf   TempC           ;save sink value in tempC
00C3 160C            0332         bsf     TempC,4         ;preset for lsd sink
00C4 0C8C            0333         rrf     TempC           ;determine next sink value
00C5 1C03            0334         btfss   STATUS,CARRY    ;c=1?
00C6 118C            0335         bcf     TempC,3         ;no then reset LSD sink
00C7 180C            0336         btfsc   TempC,0         ;else see if Msd
00C8 28D6            0337         goto    UpdateMsd       ;yes then do Msd
00C9 188C            0338         btfsc   TempC,1         ;see if 3rdLsd
00CA 28D3            0339         goto    Update3rdLsd    ;yes then do 3rd Lsd
00CB 190C            0340         btfsc   TempC,2         ;see if 2nd Lsd
00CC 28D0            0341         goto    Update2ndLsd    ;yes then do 2nd lsd
                     0342 UpdateLsd
00CD 0811            0343         movf    LsdTime,w       ;get Lsd in w
00CE 390F            0344         andlw   0x0f            ;      /
00CF 28D8            0345         goto    DisplayOut
                     0346 Update2ndLsd
00D0 0E11            0347         swapf   LsdTime,w       ;get 2nd Lsd in w
00D1 390F            0348         andlw   0x0f            ;mask rest
00D2 28D8            0349         goto    DisplayOut      ;enable display
                     0350 Update3rdLsd
00D3 0810            0351         movf    MsdTime,w       ;get 3rd Lsd in w
00D4 390F            0352         andlw   0x0f            ;mask low nibble
00D5 28D8            0353         goto    DisplayOut      ;enable display
                     0354 UpdateMsd
00D6 0E10            0355         swapf   MsdTime,w       ;get Msd in w
00D7 390F            0356         andlw   0x0f            ;mask rest
                     0357 DisplayOut
00D8 20DD            0358         call    LedTable        ;get digit output
00D9 0086            0359         movwf   PORT_B          ;drive leds
00DA 080C            0360         movf    TempC,w         ;get sink value in w
00DB 0085            0361         movwf   PORT_A
00DC 0008            0362         return
                     0363 ;
                     0364 ;
                     0365 LedTable
00DD 0782            0366         addwf   PCL             ;add to PC low
00DE 343F            0367         retlw   B'00111111'     ;led drive for 0
00DF 3406            0368         retlw   B'00000110'     ;led drive for 1
00E0 345B            0369         retlw   B'01011011'     ;led drive for 2
00E1 344F            0370         retlw   B'01001111'     ;led drive for 3
00E2 3466            0371         retlw   B'01100110'     ;led drive for 4
00E3 346D            0372         retlw   B'01101101'     ;led drive for 5
00E4 347D            0373         retlw   B'01111101'     ;led drive for 6
00E5 3407            0374         retlw   B'00000111'     ;led drive for 7
00E6 347F            0375         retlw   B'01111111'     ;led drive for 8
00E7 3467            0376         retlw   B'01100111'     ;led drive for 9
00E8 3477            0377         retlw   B'01110111'     ;led drive for A
00E9 347C            0378         retlw   B'01111100'     ;led drive for b
00EA 3439            0379         retlw   B'00111001'     ;led drive for C
00EB 345E            0380         retlw   B'01011110'     ;led drive for d
00EC 3479            0381         retlw   B'01111001'     ;led drive for E
00ED 3471            0382         retlw   B'01110001'     ;led drive for F
                     0383
                     0384 ;
                     0385 ;
                     0386 InitAd
00EE 30C0            0387         movlw   B'11000000'     ;internal rc for tad
00EF 0088            0388         movwf   ADCON0          ;      /
                     0389         ;note that adcon1 is set in InitPorts
00F0 0008            0390         return
                     0391 ;
                     0392 SampleAd
00F1 20A8            0393         call    SavePorts
00F2 20F8            0394         call    DoAd            ;do a ad conversion
                     0395 AdDone
```


© 1994 Microchip Technology Inc.

```
00F3 1908           0396          btfsc   ADCON0,GO          ;ad done?
00F4 28F3           0397          goto    AdDone             ;no then loop
00F5 1612           0398          bsf     ADOver             ;set a/d over flag
00F6 20B5           0399          call    RestorePorts       ;restore ports
00F7 0008           0400          return
                    0401 ;
                    0402 ;
                    0403 DoAd
00F8 0186           0404          clrf    PORT_B             ;turn off leds
00F9 1683           0405          bsf     STATUS,RP0         ;select pg 1
00FA 300F           0406          movlw   0x0f               ;make port a hi-Z
00FB 0085           0407          movwf   TRISA              ;       /
00FC 1283           0408          bcf     STATUS,RP0         ;select pg 0
00FD 1408           0409          bsf     ADCON0,ADON        ;start a/d
00FE 307D           0410          movlw   .125
00FF 2102           0411          call    Wait
0100 1508           0412          bsf     ADCON0,GO          ;start conversion
0101 0008           0413          return
                    0414 ;
                    0415 ;
                    0416 Wait
0102 008C           0417          movwf   TempC              ;store in temp
                    0418 Next
0103 0B8C           0419          decfsz  TempC
0104 2903           0420          goto    Next
0105 0008           0421          return
                    0422
                    0423 ;
                    0424 ;
0026                0425 count    equ     26
0027                0426 temp     equ     27
                    0427 ;
0020                0428 H_byte   equ     20
0021                0429 L_byte   equ     21
0022                0430 R0       equ     22          ; RAM Assignments
0023                0431 R1       equ     23
0024                0432 R2       equ     24
                    0433 ;
                    0434 ;
0106 1003           0435 B2_BCD   bcf     STATUS,0           ; clear the carry bit
0107 3010           0436          movlw   .16
0108 00A6           0437          movwf   count
0109 01A2           0438          clrf    R0
010A 01A3           0439          clrf    R1
010B 01A4           0440          clrf    R2
010C 0DA1           0441 loop16   rlf     L_byte
010D 0DA0           0442          rlf     H_byte
010E 0DA4           0443          rlf     R2
010F 0DA3           0444          rlf     R1
0110 0DA2           0445          rlf     R0
                    0446 ;
0111 0BA6           0447          decfsz  count
0112 2914           0448          goto    adjDEC
0113 3400           0449          RETLW   0
                    0450 ;
0114 3024           0451 adjDEC   movlw   R2
0115 0084           0452          movwf   FSR
0116 211E           0453          call    adjBCD
                    0454 ;
0117 3023           0455          movlw   R1
0118 0084           0456          movwf   FSR
0119 211E           0457          call    adjBCD
                    0458 ;
011A 3022           0459          movlw   R0
011B 0084           0460          movwf   FSR
011C 211E           0461          call    adjBCD
                    0462 ;
011D 290C           0463          goto    loop16
```


© 1994 Microchip Technology Inc.

```
                    0464 ;
011E 3003           0465 adjBCD  movlw   3
011F 0700           0466         addwf   0,W
0120 00A7           0467         movwf   temp
0121 19A7           0468         btfsc   temp,3          ; test if result > 7
0122 0080           0469         movwf   0
0123 3030           0470         movlw   30
0124 0700           0471         addwf   0,W
0125 00A7           0472         movwf   temp
0126 1BA7           0473         btfsc   temp,7          ; test if result > 7
0127 0080           0474         movwf   0               ; save as MSD
0128 3400           0475         RETLW   0
                    0476 ;
                    0477 ;
                    0478 ;
                    0479 ;
                    0480
                    0481         end
                    0482 ;
                    0483
                    0484
                    0485


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XXXXXXXXX--- -------
0140 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```

NOTES:

© 1994 Microchip Technology Inc.

AN591

Apple® Desktop Bus (ADB™)

Author: Rob McCall - WFT Electronics

INTRODUCTION

The purpose of this application note is to introduce a PIC16CXX based ADB interface which can be used as a basis for the development of custom ADB devices. This application note describes the hardware involved, a general purpose ADB protocol handler and an example application task. The example software application supports a single key keyboard to the Macintosh® computer (see Figure 1).

OVERVIEW

ADB licensing from Apple Computer.

Described as a peripheral bus used on almost all Macintoshs (except for the Macintosh 128, 512k, and Plus) for keyboards, mice, etc.

Communication between the ADB task and the application task takes place using several flags. The flags indicate whether there is data received that needs to be sent to the Macintosh, or if data from the Macintosh needs to be sent by the application.

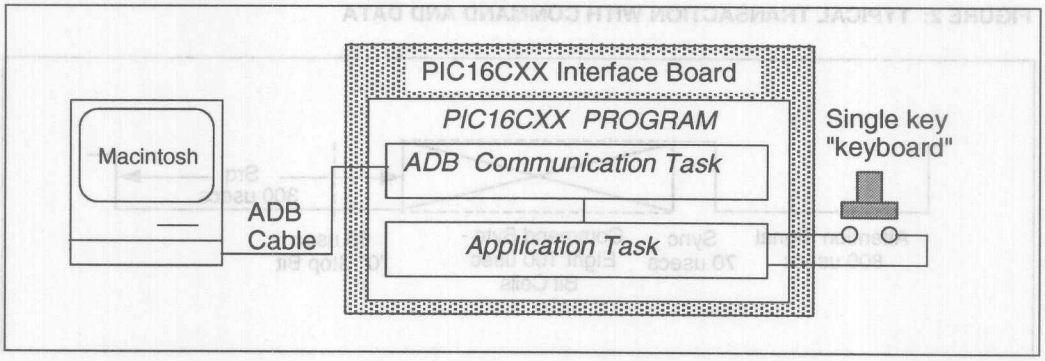
EXPLANATION OF ADB TECHNOLOGY

The ADB is an asynchronous pulse-width communication protocol supporting a limited number of devices. All devices share a single I/O wire in a multi-drop master/slave configuration in which any slave devices may request service. This accomplished through a wired OR negative logic arrangement.

The ADB cable is composed of four wires: +5v, gnd, ADB signal, and power-on (of the Macintosh). The signal wire communicates ADB input and output using an open collector type signal. The number of devices is limited by the addressing scheme and a maximum current draw of 500mA.

Every ADB device has a default address at start-up assigned by Apple. If there are device address conflicts, the protocol supports the reassignment of device addresses at start-up. The software in the PIC16CXX discussed here is designed to easily modify the device address to make the PIC16CXX appear as another ADB device for testing and development.

FIGURE 1: BLOCK DIAGRAM OF FUNCTIONALITY



© 1994 Microchip Technology Inc.

No device issues commands, except the host. However, devices are permitted to request service during specific time intervals in the signal/Command protocol. A Service Request is referred to as an "**Srq**" The signal protocol communication is accomplished by pulling the ADB line low for various time intervals.

The host controls the flow of data through issuance of specific signal sequences and by issuing several types of Commands. The basic Command types are Talk, Listen, Flush, or Reserved. Each Command has a component called a "Register" indicator which specifies the storage area affected by the Command type. The following is a summary explanation of the each of the Commands. The complete specifications are available from Apple, as listed in the Resources section.

PROTOCOL ASSUMPTIONS

The ADB protocol is defined with a number of general assumptions about its use. These assumptions have driven the general philosophy of the communication sequences. It is assumed that the devices on the ADB are used for human input and each are used one at a time, such as a keyboard and a mouse. It is also assumed that the transfer time from one device to another is relatively slow. This does not mean that the protocol is limited to these assumptions but rather that the protocol is optimized towards this type of use. This is made very evident in the host polling logic, where the host continues to poll the last device communicated with until another device issues an Srq. Consequently, if another device issues an Srq, the device being communication with (or the host) may need to retransmit.

ADB Elements:

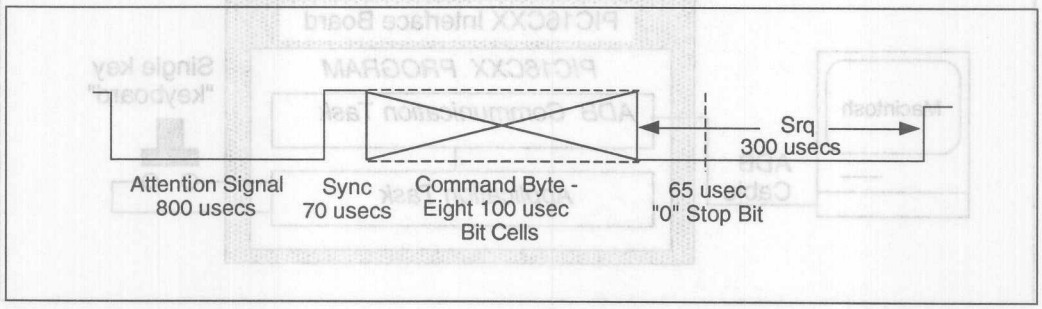
The ADB protocol has two components, a signal protocol and a Command/data protocol. These two elements are intertwined. The signal protocol is differentiated in most cases by timing periods during which the ADB signal is low. The Apple ADB specification allows +/- 3% tolerance timing of the signals from the host and +/- 30% by the devices. The signals are:

- Reset: signal low for 3 ms.
- Attention: signal low for 800 µs.
- Sync: signal high for 70 µs.
- Stop-to-Start-Time (**Tlt**): signal high for 160 to 240 µs.
- Service Request (**Srq**): signal low for 300 µs.

After device initialization, in general all communication through the ADB is accomplished through the following event sequence initiated by the host: **Attention** Signal, **Sync** Signal, **Command Packet**, **Tlt** signal, and **Data Packet** transfer. Depending upon the Command, the device may or may not respond with a data packet. Service Requests are issued by the devices during a very specific time at the end the reception of the Command packet.

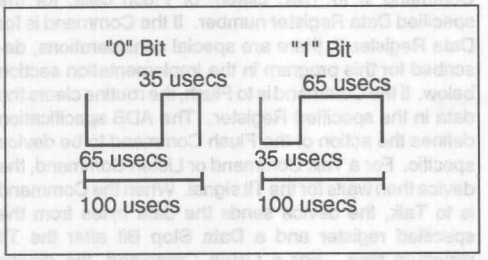
The Command packets and the Data packets are the constructs used to communicate the digital information. The method of representing data bits is accomplished in a signal timing construct called a **bit cell**. Each **bit cell** is a 100 µs period. Data 1's and 0's are defined by the proportions of the bit cell time period when the line is low and then high. A "1" bit is represented by the line low for 35 µs, and high for 65 µs. Conversely, A "0" bit is represented by the line low for 35 µs, and high for 65 µs (see Figure 3).

FIGURE 2: TYPICAL TRANSACTION WITH COMMAND AND DATA



© 1994 Microchip Technology Inc.

FIGURE 3: BIT CELLS



The **Command Packet**, received from the host, follows an Attention signal and a Sync signal. It consists of an 8 bit Command Byte and a "0" Command Stop Bit. The Command Byte may be broken down into two nibbles. The upper nibble is a four bit unique Device Address. The lower nibble is defined as a Global or Reserved Command for all devices, or a Talk, Listen, or Flush Command for a specific device. Also contained in the lower nibble is a "Register" designator which further details the Command. The importance of the Command Stop bit cell is that **Srq**'s can only be issued by a device to the host during the Command Stop Bit cell low time if the device address is not for the device wishing service, The Host controls when Srq's are allowed through the Command protocol. The Tlt signal and Data Packet transfer, which are part of every Command packet signal sequence, are overridden if an Srq is issued by any device.

A **Data Packet** is the data sent to, or received from, the host. Its length is variable from 2 to 8 bytes. The structure is a "1" Start Bit, followed by 2 to 8 bytes, ending with a "0" Stop Bit. The Apple ADB documentation refers to the data packet sent or requested as Device Data "Registers". This does not necessarily indicate a specific place in memory. In this PIC16CXX implementation, each Data Register has been limited to two PIC16CXX register bytes. The ADB specification allows each Data Register to hold between two and eight bytes. They are referenced in the Command byte as "register" as 0, 1, 2, or 3. Data Register 3 has special significance. It holds the special status information bits (such as whether Srq's are allowed), the Device Address, and the Device Handler ID. Commands are further defined by the "register id" sent in the Command data packet.

For example, if the Host issues the Command in binary of 0010 1100, it would be interpreted as "Device 2, Talk Register 0". The complete definition of the Commands and data registers are described in detail in the ADB specifications supplied by Apple.

PIC16CXX ADB PROTOCOL PROGRAM EVENT SEQUENCE:

Overview

At power-on the host will generate a Reset signal. The purpose of Reset is to initialize the devices on the ADB line. This includes determining the addresses of each device, and resolving device address conflicts if there are any. Once the device addresses are determined, each device waits to be Commanded or issues an Srq if it requires service from the host and is not being addressed by the host. After Reset processing the ADB Protocol Task monitors the ADB line for the Attention/Sync/Command signal sequence. The PIC16CXX program differentiates the signal timing.

Note: The signal detection routines check to see if the Application Task needs service after each event and after the falling edge of the Attention signal is detected.

Command interpretation is accomplished during the low signal time of the Stop Bit cell of the Command packet. Response to the Command must occur after the minimum time of the Stop to Start time period (**Tlt**), which is 160 usec. but before the max Tlt time of 240 usecs. When a device has issued an Srq, it waits to be addressed by the host. If the next Command received is not for the device, it issues the Srq again. The normal response to an Srq will be a Talk Command from the host.

Detailed Description

START-UP

Upon start-up, the Reset routine is executed, looking for the ADB line to be high. When the line is high, an initialization routine is executed during which registers are cleared or loaded with default values. The only exception is a register for generating a random address used in the address conflict resolution process.

RESET

During a Reset condition, default values are loaded, such as the Default Device Address and Handler ID (a piece of information used by the host to identify the type of device). More than one device may have the same address. There is a sequence of events to resolve address conflicts described in the Implementation section. The host assigns a unique address to each device. The Reset condition only takes place once, during start-up, except under unusual conditions, such as testing this program.

3

© 1994 Microchip Technology Inc.

ATTENTION ROUTINE

When the Reset routine is complete, the Attention Signal routine is executed, looking for the line to go low and then high. This low time is monitored to be within range of the Attention Signal Timing. If the timing is below the minimum threshold, the routine aborts to start over again looking for the line to go low as the beginning of the Attention Signal. If the low time is exceeded, the routine aborts to the Reset Signal routine.

SYNC SIGNAL ROUTINE

When the line transitions to high, the Sync Signal routine looks for the line to go low as the start of the first bit of the Command Byte. If the Sync high time is exceeded, the routine aborts to the Attention Signal.

COMMAND ROUTINE

The Command routine detects and decodes the next 8 bit cells as the Command Byte. The routine must first determine if the device address is for the device. If the routine determines that the device address in the Command matches the stored device address it may do one of two things; issue an Srq to the host by holding the line low, or go on to check if the Command is Global to all devices. If Global, the routine determines the specific Command and executes the routine for that Global Command. After execution of the Command routine it then goes back to look for the Attention Signal.

When a device is addressed, it determines whether the Command is to Talk, Listen, or Flush data, for the specified Data Register number. If the Command is for Data Register 3, there are special considerations, described for this program in the Implementation section below. If the Command is to Flush, the routine clears the data in the specified Register. The ADB specification defines the action of the Flush Command to be device specific. For a Talk Command or Listen Command, the device then waits for the Tlt signal. When the Command is to Talk, the device sends the data bytes from the specified register and a Data Stop Bit after the Tlt minimum time. For a Listen Command, the device receives data for the specified Register.

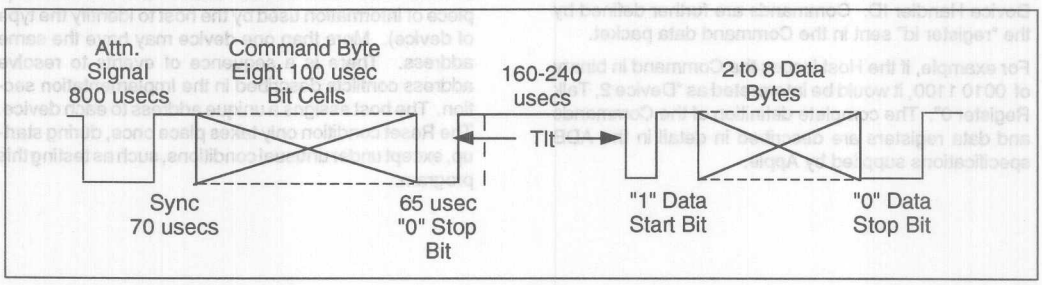
When the data has been Flushed, Sent, or Received, the program the device then goes to look for the Attention signal again.

Notes:
1. In this PIC16CXX program, the Application Task is serviced before looking for the Attention signal.
2. If at any time the line is low or high outside of timing ranges, the program aborts to check if an Attention or Reset signal has been issued by the Host. In the case of sending Data, the program goes first to the Collision routine.

SENDING DATA TO THE HOST

Data is sent only in response to a Talk Command. For every data bit cell, the line is tested to go high at the proper time. If the line is still low, a collision has occurred. When a collision is detected, a collision flag is set, and the program aborts to look for a Command signal sequence.

FIGURE 4: TYPICAL TRANSACTION WITH SERVICE REQUEST



© 1994 Microchip Technology Inc.

IMPLEMENTATION

Hardware:

The hardware of this circuit is fairly simple. The circuit is powered via the +5v and Gnd wires of the ADB cable. The ADB I/O wire is connected to port RA0 with a pull-up resistor to 5v. The T0CKI pin is tied to Gnd. The Master Clear pin is tied to 5v.

This circuit uses a 4Mhz crystal as a timing reference, but higher values may be substituted. The software is designed to accommodate higher frequencies.

A push-button switch is used as the single key of the "keyboard". One side is connected to port RB1 with a pull-up resistor to 5v, and the other side to Gnd. An LED is used to indicate that the 'key' has been pressed, with the positive side connected to port RB0 and the negative side to Gnd.

Software:

There are two sections of the program designated as "Application Tasks", setup to switch between a protocol support task for the ADB signal decode and processing, and another task, the Application Task, in this case a single key "keyboard" routine. The ADB protocol task has priority. The first section of the code is the ADB protocol task, the second section is the Application task, "Keyboard". The two tasks communicate through flags which indicate that data needs to be sent, or that data has been received.

The Keyboard Task is run at two times; 1) during the Attention Signal, 2) between the end of the Data Stop Bit and the beginning of the Attention Signal. The Keyboard Tasks is given up to 500 usecs during the Attention Signal, and 900 usecs during the time between the end of the Data Stop Bit and the beginning of the Attention Signal. It is important to note here that the other tasks MUST NOT AFFECT TMR0 or the ADB time variable that the Attention Signal is using to keep track of the RTCC.

Timing:

Timing of is accomplished by first loading a constant into a time variable. This constant represents the maximum limit for the current routine, which may not necessarily be the maximum timing range for the current Signal. The TMR0 value is loaded into the working register, and subtracted from the time variable. The Carry bit of the Status register is tested to see if it is set or cleared. If the bit is clear, the current timing limit has been exceeded. Further action is taken based on this status. It is important that the constant not be allowed very close to 255, or rollover may occur, giving inaccurate results. The prescaler is applied to the TMR0 as necessary.

The following are the timing ranges used by this program for ADB signals:

Reset	Greater Than 824 usecs
Attention	776-824 usecs
Sync	72 usecs
Bit Cell	Up to 104 usecs
1 Bit low time	< 50 usecs
0 Bit low time	> 50 < 72 usecs
Stop bit	0 Bit
Stop to Start (Tlt)	140-260 usecs
Service Request (Srq)	300 usecs

Note: The range of values given for 0 bit, 1 bit, and Tlt timing are slightly wider than those given in the ADB specification.

How Address Conflicts are Resolved:

During the start-up process the host sends a "Talk Register 3" Command to each device address, and waits for a response. When a device recognizes that the Host issued a "Talk Register 3" Command, it responds by sending a random address. During the transfer of each bit cell of the random address the signal line is monitored for the expected signal level. If the signal is not what is expected there is an address conflict. If the address is sent successfully the host will respond with a Listen Command to that device with a new Device Address for that device to move to. The device then only responds to Commands at the new Address.

If there is a conflict, where two devices have the same default address, and respond at the same time, the device that finds the line low when it expects it to be high, immediately stops transmitting because it has determined that a collision has occurred. The device which detected the collision marks its address as unmovable and therefore ignores the address move Command, a Listen register 3 Command, The device maintains the unmovable address condition until it has executed a successful response to the talk register 3 Command.

The host continues sending a Talk Register 3 Command at the same address until there is a time-out and no device responds. This is how conflicts are resolved when more than one device has the same address; for example, if two keyboards are connected.

© 1994 Microchip Technology Inc.

Program Sequence:

Words in parenthesis accompanying the TITLES are Labels of procedures corresponding in the code.

Start-up / IDLE (Start)

Start by setting the ADB pin on Port A and the Switch Pin on Port B as inputs, and tri-stating the rest of Port A and B as outputs.

INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH (Reset)

Look for the line to be high, and when it is, clear or initialize registers to default values.

LOOK FOR ATTENTION OR RESET (AttnSig)

Look for the line to go low, when it does, clear TMR0 and time how long it is low. An Attention Signal has occurred when the line goes high between 776 and 824 usecs. If the low time is measured less than 776 usecs, another signal has occurred and the program aborts, looking for the Attention Signal again. When the low time is measured greater than 824 usecs, the program interprets this timing as a Reset Signal. The program starts over again, waiting for the line to be high, and when it is, performs a Reset initialization.

Note: The keyboard task is performed during the attention signal (Task_2).

LOOK FOR SYNC SIGNAL (SyncSig; calls Srq)

The Sync Signal is the high time between the rising edge of the Attention Signal and the falling edge of the first bit of the Command.

GET THE COMMAND (Command; calls Get_Bit)

Look for the Command; a combination of eight 0 and 1 bits. The MSB is sent first. This is achieved by calling a the Get_Bit routine, which checks whether the maximum Bit Cell time is exceeded, if not, it looks for the rising edge at the end of the bit. When the bit is received, it is rotated into a variable, and the end of the bit cell is expected. When the falling edge of the next bit is detected, the routine clears TMR0 and returns to Command, which calls Get_Bit again until all 8-bits of the Command have been received.

ISSUE A SERVICE REQUEST IF NECESSARY (Srq)

If data needs to be sent to the Host, a Service Request (Srq) is issued by holding the line low while the Stop Bit is being received, during the Stop-to-Start time (Tlt) between the end of the Command Stop bit and the beginning of the Data Start Bit

LOOK FOR STOP BIT (CmdStop)

Look for the Stop Bit (a 0 bit of 65 usecs) that comes after the last Command Byte.

INTERPRET THE COMMAND (AddrChk)

After the Command has been received, determine if the Address belongs to this Device. If the Address is not for this Device, determine if the Command is global for all Devices and if so, do that Command. If this is not a Global/Reserved Command, call the Service Request (Srq) routine to see if an Srq should to be issued to the Host, and do so if necessary, then return to get the Attn Signal. If the Address is for this Device determine whether it is a Talk, Listen, or Flush Command, and go to the specified Command routine.

SENDING DATA (Talk; calls Tlt)

If the Command was interpreted to be a Talk Command addressed to this Device, call the Stop-to-Start Time (Tlt) routine. When the Tlt routine has completed, determine if this is a Talk Register 3 Command. If so, return a Random Address as part of the two bytes sent to the Host. If this is not a Talk Register 3 Command, determine if Data needs to be sent. If so, send the Data Start Bit (a '1'), two bytes of Data from the indicated register, and a Stop Bit (a '0'). If not, abort to the Attention Signal. If at any time the transmission of Data is interrupted, abort to the Collision routine. Only after a complete transmission should the flags be cleared indicating a successful transmission.

Note: The ADB Specification indicates data may be between two and eight bytes long. The limitations of the PIC16C54/55/56 parts allow only two bytes of data to be sent by this program due to limited register space. If more than two bytes of data must be sent, use the PIC16C57.

RECEIVING DATA (Listen; calls Tlt)

If the Command was interpreted to be a Listen Command addressed to this Device, call the Stop-to-Start Time (Tlt) routine. When the Tlt routine has completed, receive the rest of the Data Start Bit, 2 Data Bytes, and Data Stop Bit. When the Data has been received, determine whether this is a Listen Register 3 Command. If this is a Listen Register 3 Command, interpret what the Command is. If this is a conditional Address change Command, determine if this Device's Address is moveable at this time. If not, abort to the Attention Signal. If so, change the Device to the new Address and go run the Second Application Task. If this is not a Listen Register 3 Command, move the Data into the specified register and go run the Second Application Task.

© 1994 Microchip Technology Inc.

LOOK FOR THE STOP TO START TIME (Tlt)

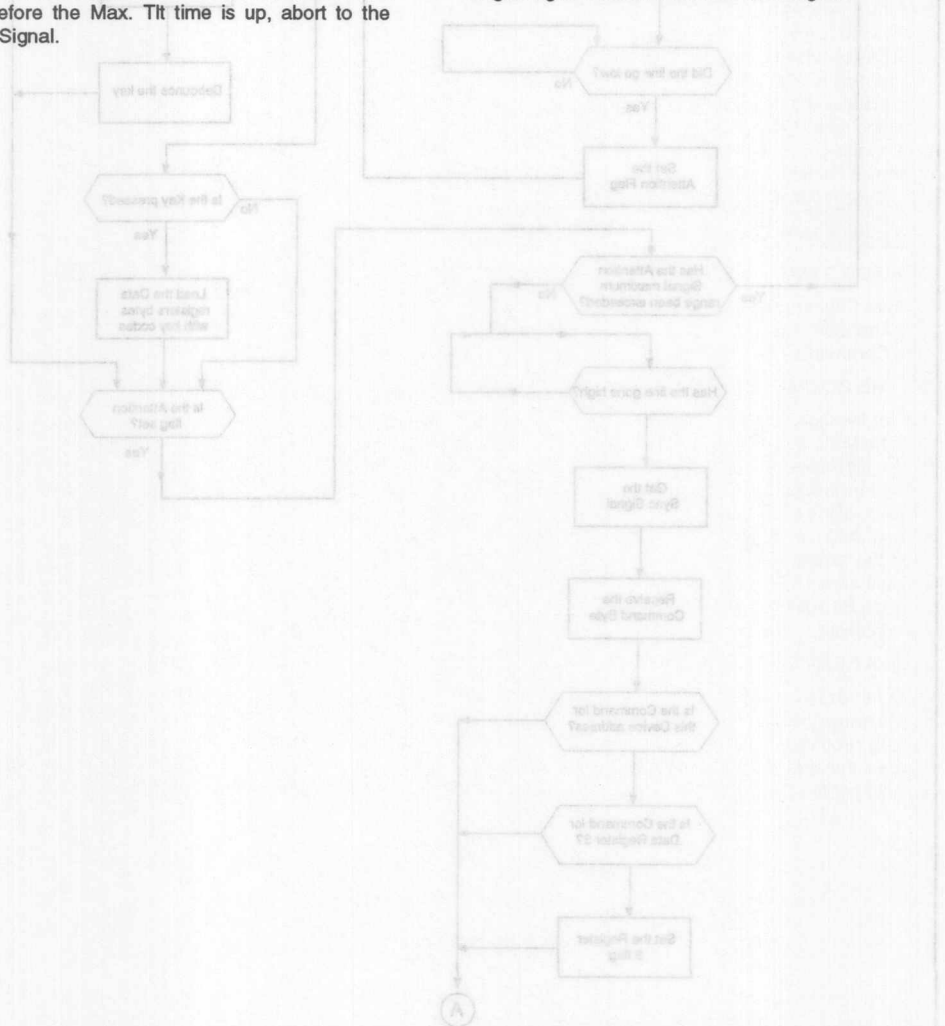
After the Command and Stop Bit, the Talk or Listen routines call the Tlt routine. Tlt looks for the line to go low. If the line went low before the Min. Tlt Time, see if this is a Talk Command. If this is a Talk Command, abort to the Collision routine. If this is a Listen Command, abort to the Attention Signal.

If the Min. Tlt time passes and the line is high, see if the Talk routine called the Tlt, if so, go wait for until the middle of the Tlt, then return to the Talk routine to send the Data Start Bit, Data Bytes, and Stop Bit. If at any time the line goes low during the Tlt and the Talk routine called it, abort to the Collision routine.

If the Listen routine called Tlt, look for the line to go low as the beginning of the Data Start Bit. When the line goes low, return for the rest of the Start Bit. If the line doesn't go low before the Max. Tlt time is up, abort to the Attention Signal.

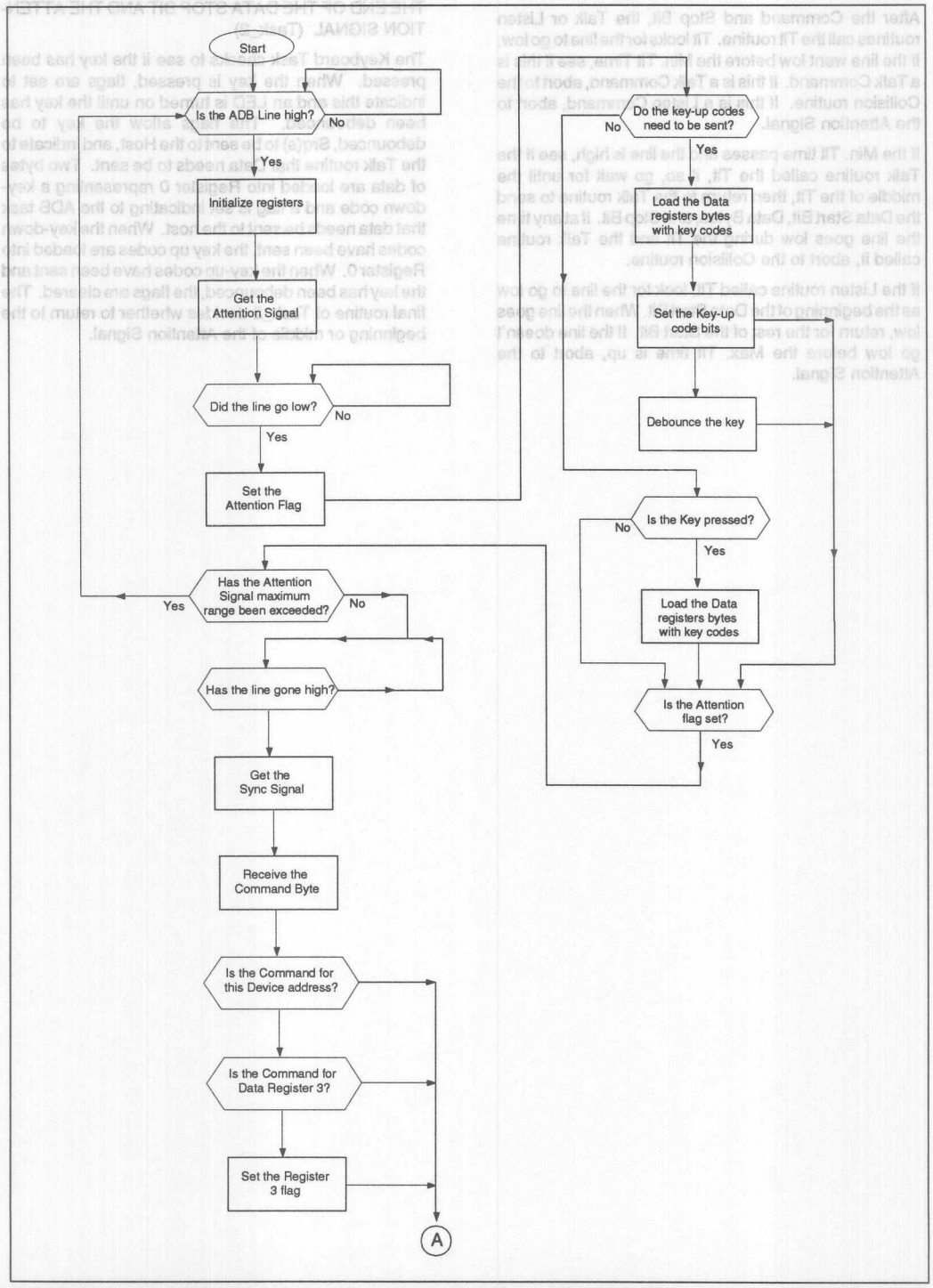
THE KEYBOARD TASK IS PERFORMED BETWEEN THE END OF THE DATA STOP BIT AND THE ATTENTION SIGNAL (Task_2)

The Keyboard Task checks to see if the key has been pressed. When the key is pressed, flags are set to indicate this and an LED is turned on until the key has been debounced. The flags allow the key to be debounced, Srq(s) to be sent to the Host, and indicate to the Talk routine that Data needs to be sent. Two bytes of data are loaded into Register 0 representing a key-down code and a flag is set indicating to the ADB task that data needs be sent to the host. When the key-down codes have been sent, the key up codes are loaded into Register 0. When the key-up codes have been sent and the key has been debounced, the flags are cleared. The final routine of Task_2 decides whether to return to the beginning or middle of the Attention Signal.



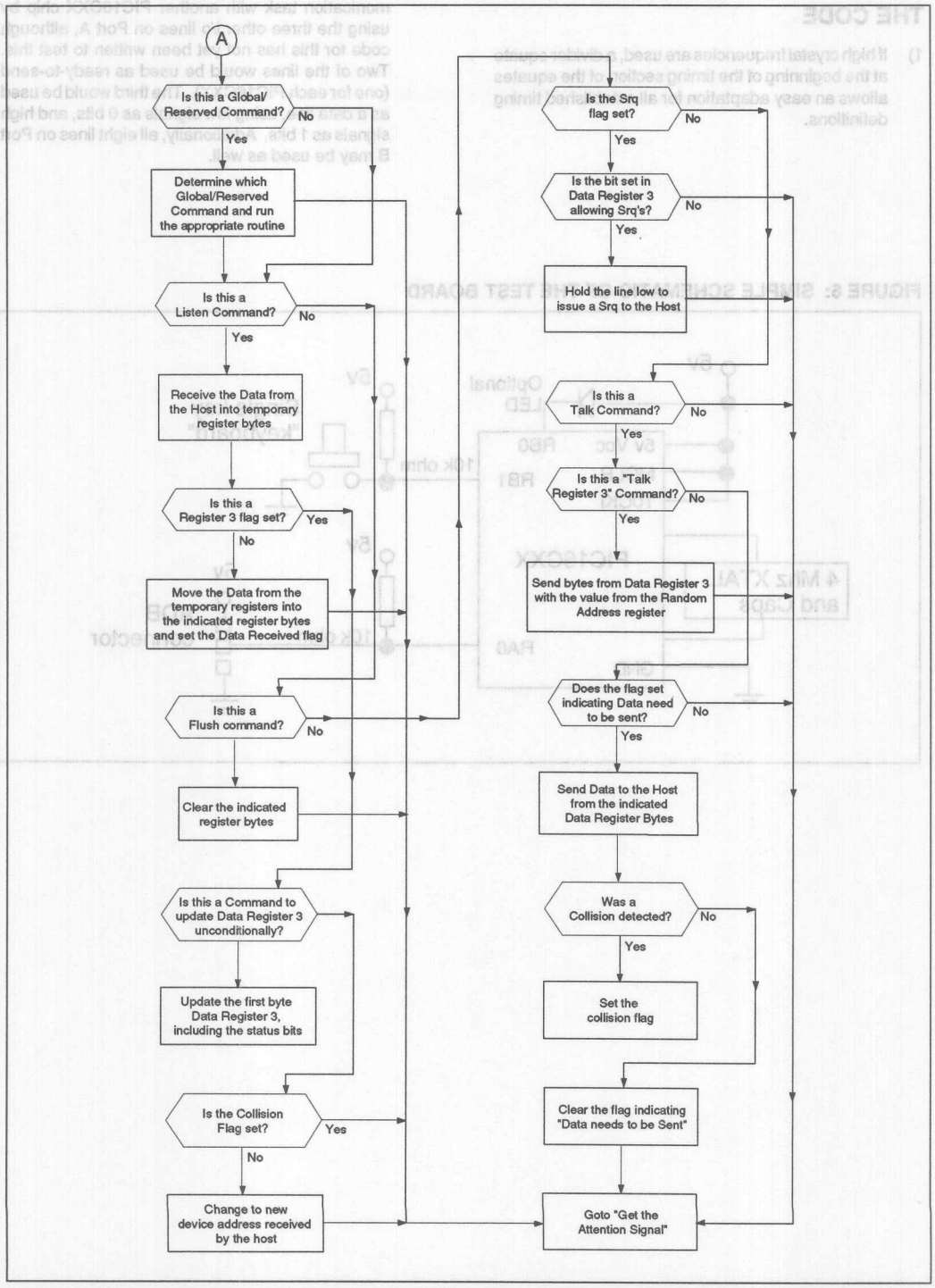
© 1994 Microchip Technology Inc.

FIGURE 5: APPLE DESKTOP BUS PIC16CXX FLOWCHART



© 1994 Microchip Technology Inc.

FIGURE 5 (CONT.): APPLE DESKTOP BUS PIC16CXX FLOWCHART



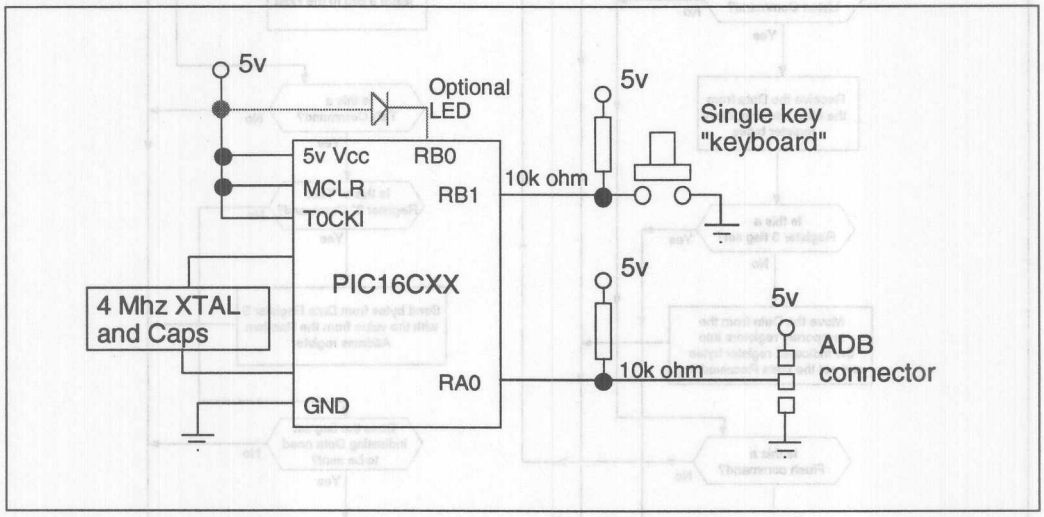
© 1994 Microchip Technology Inc.

SUGGESTIONS ABOUT MODIFYING THE CODE

1) If high crystal frequencies are used, a divider equate at the beginning of the timing section of the equates allows an easy adaptation for all established timing definitions.

2) The second application task may occur as a communication task with another PIC16CXX chip by using the three other i/o lines on Port A, although code for this has not yet been written to test this. Two of the lines would be used as ready-to-send (one for each PIC16CXX). The third would be used as a data line, using low signals as 0 bits, and high signals as 1 bits. Additionally, all eight lines on Port B may be used as well.

FIGURE 6: SIMPLE SCHEMATIC OF THE TEST BOARD



© 1994 Microchip Technology Inc.

RESOURCES

Apple Publications and Support Software

MacTech Magazine (formerly MacTutor) is a publication dedicated to supporting the Macintosh. They have had several articles regarding the Apple Desktop Bus. They publish a CD-ROM that contains all of their articles from 1984 to 1992. Also, single disks are available (ask for #42).

MacTech Magazine can be contacted at:

P.O. Box 250055
Los Angeles, CA 90025-9555
310 575-4343 FAX 310 575-0925
Applelink: MACTECHMAG
Internet: info@xplain.com

Apple licenses the ADB technology. They can be contacted at:

20525 Mariani Ave.
Cupertino, CA 95014
Attn: Software Licensing

- Apple Keyboard, extended, specification drawing #062-0168-A.
- Apple Desktop specification drawing # 062-0267-E.
- Apple Desktop connector, plug, Mini DIN drawing #519-032X-A.
- Engineering Specification, Macintosh transceiver interface, ADB drawing #062-2012-A.
- Apple keyboard, specification drawing #062-0169-A.
- Developer CD series, Tool Chest Edition, August 1993 contains:
  - Folder = Tool Chest: Devices and Hardware: Apple Desktop Bus
  - ADB Analyzer
  - ADB Parser (most complete environment)
  - ADB Lister
  - ADB ReInit
  - ADB Tablet code samples

WFT Electronics offers free assistance in procuring necessary ADB info. Contact Gus Calabrese, Rob McCall, Dave Evink at:

4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.orgA

AUTHOR / CREDITS

Rob McCall developed the majority of the PIC16CXX ADB code. He also wrote most of the application note. Gus Calabrese, Dave Evink, and Curt Apperson supported this effort. Dave works with Gus, Rob, and Curt in developing a variety of embedded processor products.

Contact Gus Calabrese, Rob McCall, Dave Evink, Curt Apperson at:

WFT Electronics
4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.org

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN555

Software Implementation of Asynchronous Serial I/O

INTRODUCTION

The PIC16CXX microcontrollers from Microchip Technology, Inc., are mid-range, high performance EPROM based 8-bit microcontrollers. Some of the members of this series (like PIC16C71 and PIC16C84) do not have an on-chip hardware asynchronous serial port. This application note describes the Interrupt driven Software implementation of Asynchronous Serial I/O (Half Duplex RS-232 Communications) using PIC16CXX microcontrollers. These microcontrollers can operate at very high speeds with a minimum of 250 ns cycle time (with input clock frequency of 16 MHz). To test the RS-232 routines, a simple Digital Volt Meter (DVM)/Analog Data Acquisition Systems has been implemented using PIC16C71 in which upon reception of a command from host (IBM® PC), an 8-bit value of the selected A/D channel is transmitted back to host.

IMPLEMENTATION

A half duplex Interrupt driven software implementation of RS-232 communications using PIC16C71 is described in detail below. The transmit pin used in the example code is RB7 and receive pin is connected to RTCC/RA4 pin (see Figure 2). Of course these pins are connected with appropriate voltage translation to/from RS-232/CMOS levels. The voltage translation is given described with schematics in the hardware section of this application note.

Transmit Mode

The transmit mode in software is quite straight forward to implement using interrupts. Once the input clock frequency and baud rate is known, the number of clock cycles per bit can be computed. The on-chip Real Time Clock Counter (RTCC) along with the prescaler can be used to generate interrupt on RTCC overflow. This RTCC overflow interrupt can be used as timing to send each bit. The Input clock frequency ("_ClkIn") and the Baud Rate ("_BaudRate") are programmable by the user and the RTCC time-out value (the period for each bit) is computed at assembly time. Whether the prescaler must be assigned to RTCC or not is also determined at assembly time. This computation is done in the header file "rs232.h". Note that very high speed transmissions can be obtained if transmission is done with software delays instead of every interrupt driven, however, the processor will be totally dedicated to this job.

Transmission of a byte is performed by calling "PutChar" function and the data byte in the "TxReg" is transmitted out. Before calling this function ("PutChar"), the data must be loaded into TxReg and also made sure that serial port is free. The serial port is free when both _txmtProgress and _rcvOver bits are cleared (see description of these bits in the Serial Status/Control Reg table given later).

Summary of "PutChar" function :

1) Make sure _txmtProgress & _rcvOver bits are cleared
2) Load TxReg with data to be transmitted
3) CALL PutChar function

Receive Mode

The reception mode implementation is slightly different from the transmit mode. Unlike the transmit Pin (TX pin in the example code is RB7, but could be any I/O pin), the receive pin (RX Pin) must be connected to RTCC/RA4 Pin. This is because in reception, the Start Bit which is asynchronous in nature, must be detected. To detect the start bit, when put in Reception mode, the RTCC module is configured to counter mode . The OPTION register is configured so that RTCC module is put in counter mode (increment on external clock on RTCC/RA4 Pin) and set to increment on falling edge on RTCC/RA4 pin with no prescaler assigned. After this configuration setup, RTCC (File Reg 1) is loaded with 0xFF. A falling edge on RTCC Pin will make RTCC roll over from 0xFF to 0x00, thus generating an interrupt indicating a Start Bit. The RTCC/RA4 pin is sampled again to make sure the transition on RTCC is not a glitch. Once the start bit has been detected, the RTCC module is reconfigured to increment on internal clock and the prescaler is assigned to it depending on input master clock frequency and the baud rate (configured same way as the transmission mode).

The software serial port is put in reception mode when a call is made to function "GetChar". Before calling this function make sure serial port is free (i.e. _txmtProgress and _rcvOver status bits must be 0). On completion of reception of a byte, the data is stored in RxReg and _rcvOver bit is set to 0.

Summary of "GetChar" function:

1) Make sure _txmtProgress & _rcvOver bits are cleared
2) CALL GetChar function
3) The received Byte is in TxReg after _rcvOver bit is cleared

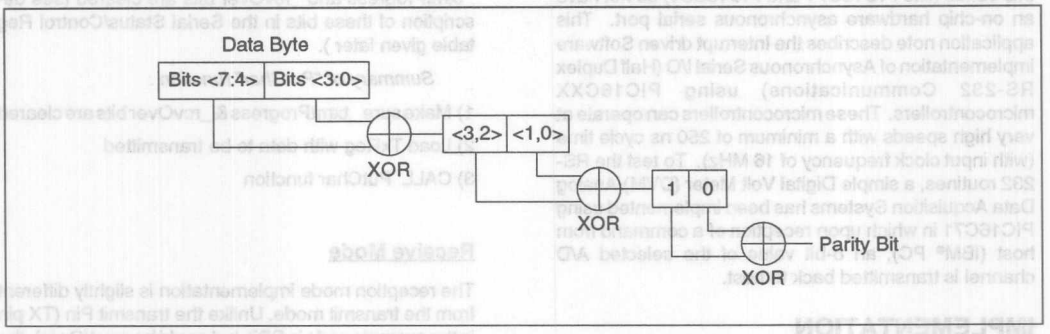
3

© 1994 Microchip Technology Inc.

Parity Generation

Parity can be enabled at assembly time by setting "_PARITY_ENABLE" flag to TRUE. If enabled, the parity can be set to either EVEN or ODD parity. In transmission mode, if parity is enabled, the parity bit is computed and transmitted as the ninth bit. On reception, the parity is computed on the received byte and compared to the ninth bit received. If a match does not occur the parity error bit is set in the RS-232 Status/Control Register (_ParityErr bit of SerialStatus reg). The parity bit is computed using the algorithm shown in Figure 1. This algorithm is highly efficient using PIC16CXX's SWAPF and XORWF instructions (with ability to have the destination as either file register itself or W register) and the sub-routine (called "GenParity") is in file "txmtr.asm".

FIGURE 1 - AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE



Assembly Time Options

The firmware is written as a general purpose routines and the user must specify the following parameters before assembling the program. The Status/Control register is also described below:

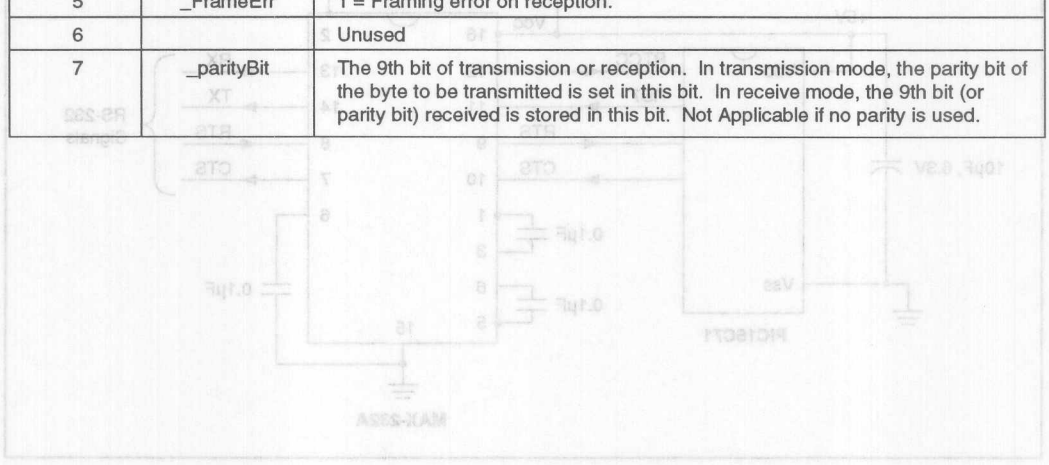
TABLE 1 - LIST OF ASSEMBLY TIME OPTIONS

_ClkIn	Input clock frequency of the processor.
_BaudRate	Desired Baud Rate. Any valid value can be used. The highest Baud Rate achievable depends on Input Clock Freq. 600 to 4800 Baud was tested using 4 MHz Input Clock. 600 to 19200 Baud was tested using 10 MHz Input Clock. Higher rates can be obtained using higher Input Clock Frequencies. Once the _BaudRate & _ClkIn are specified, the program automatically selects all the appropriate timings.
_DataBits	Can specify 1 to 8 data bits.
_StopBits	Limited to 1 Stop Bit. Must be set to 1.
_PARITY_ENABLE	Parity Enable Flag. Set it to TRUE or FALSE. If PARITY is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag description below.
_ODD_PARITY	Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else mEVEN Parity Scheme is used. This Flag is ignored if _PARITY_ENABLE is set to FALSE.
_USE_RTSCTS	RTS & CTS Hardware handshaking signals. If set to FALSE, no hardware handshaking is used. If set to TRUE, RTS & CTS use up 2 I/O Pins of PortB.

© 1994 Microchip Technology Inc.

TABLE 2 - BIT ASSIGNMENTS OF SERIAL STATUS/CONTROL REGISTER ("SERIALSTATUS" REG)

Bit #	Name	Description
0	_txmtProgress	1 = Transmission in progress. 0 = Transmission line free.
1	_txmtEnable	Set this bit to 1 on initialization to enable transmission. This bit may be used to abort a transmission. The transmission is aborted if in the middle of a transmission (i.e. when _txmtProgress bit is 1) _txmtEnable bit is set to 0. This bit gets automatically set when PutChar function is called.
2	_rcvProgress	1 = Middle of a byte reception. 0 = Reception of a byte (in RxReg) is complete and is set to 1 when a valid start bit is detected in reception mode.
3	_rcvOver	0 = Completion of reception of a byte. The user's code can poll this bit after calling "GetChar" function and check to see if it is set. When set, the received byte is in RxReg. Other status bits should also be checked for any reception errors.
4	_ParityErr	1 = Parity error on reception (irrespective of Even Or Odd parity chosen). Not applicable if No Parity is used.
5	_FrameErr	1 = Framing error on reception.
6		Unused
7	_parityBit	The 9th bit of transmission or reception. In transmission mode, the parity bit of the byte to be transmitted is set in this bit. In receive mode, the 9th bit (or parity bit) received is stored in this bit. Not Applicable if no parity is used.



Hardware

The hardware is primarily concerned with voltage translation from RS-232 to CMOS levels and vice versa. Three circuits are given below and the user may choose which ever best suits his application. The primary difference between each solution is cost versus number of components. Circuits in Figure 3 and 4 are very low cost but have more components than the circuit in Figure 2. The circuit in Figure 2 interfaces to RS-232 line using a single chip (MAX-232) and single +5V supply. The circuit in Figure 3 is a low cost RS-232 Interface but requires two chips and a single +5V supply source.

Figure 4 shows a very low cost RS-232 Interface to an IBM PC® with no external power requirements. The circuit draws power from RS-232 line (DTR) and meets the spec of drawing power less than 5mA. This requires that the host to communicate must assert DTR high and RTS low. The power is drawn from DTR line and this requires that DTR to be asserted high and must be at least 7V. The negative -5 to -10 V required by LM339 is drawn from RTS line and thus the host must assert RTS low. This circuit is possible because of the low current consumption of PIC16C71 (typical 2 mA).

FIGURE 2 - SINGLE CHIP FOR RS-232 INTERFACE (SINGLE +5V SUPPLY)

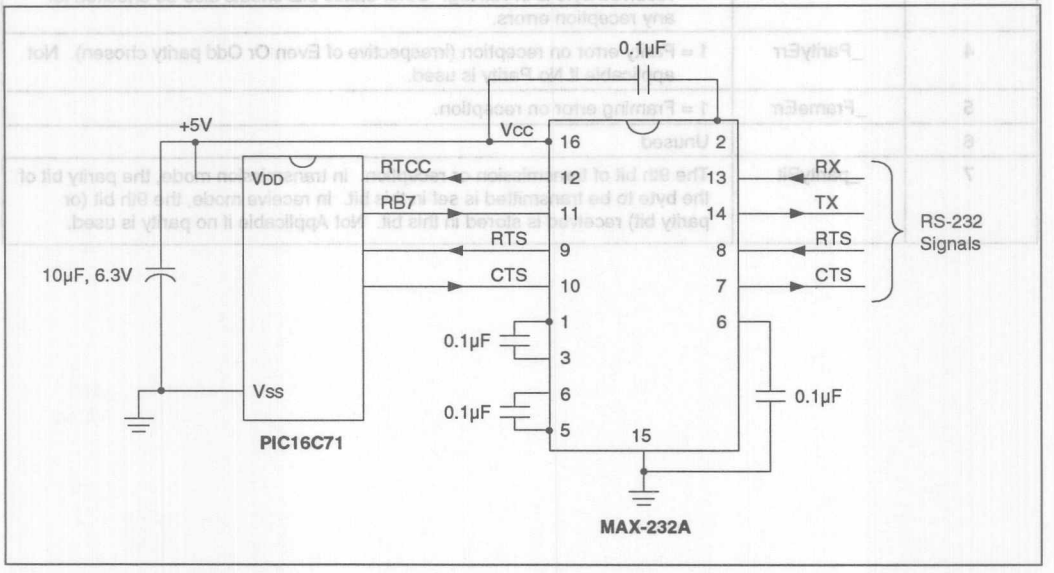
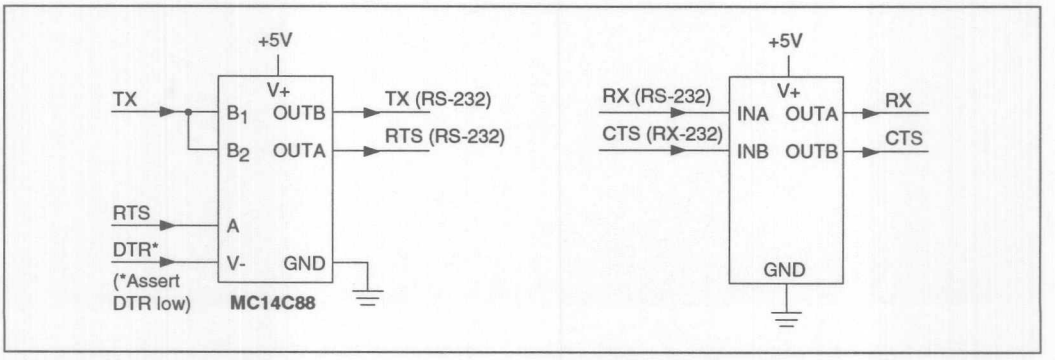
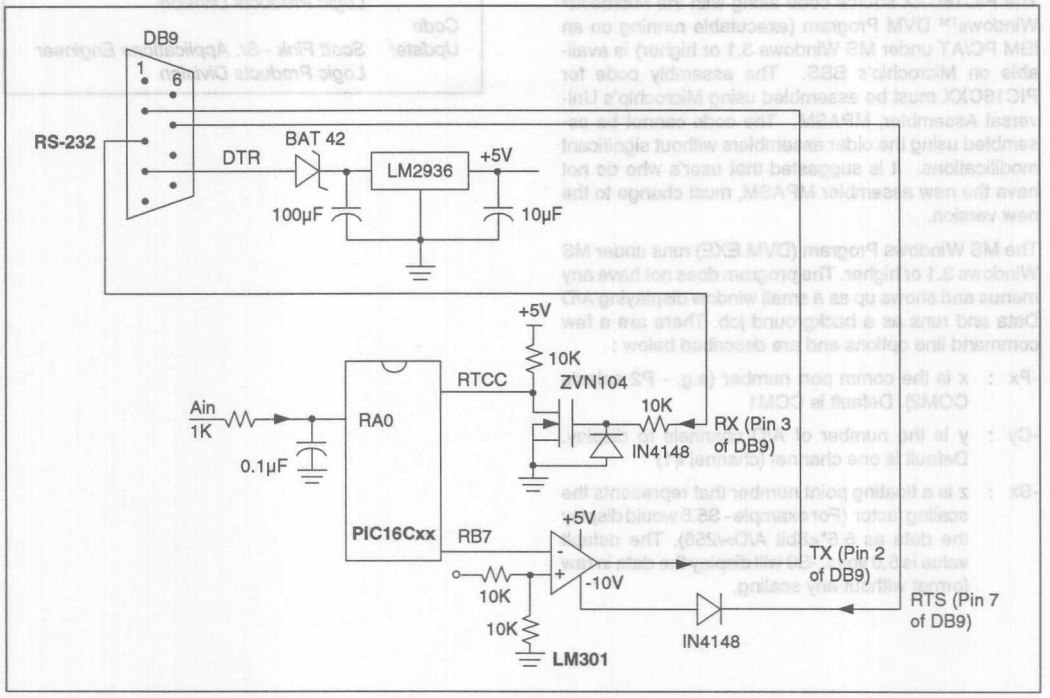


FIGURE 3 - LOW COST RS-232 INTERFACE (TWO CHIPS, SINGLE +5V SUPPLY)



© 1994 Microchip Technology Inc.

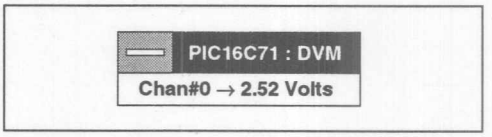
FIGURE 4 - LOW COST, LOW POWER RS-232 INTERFACE (POWER SUPPLIED BY RS-232 LINES)



Test Program

To test the transmission and reception modules, a main program is written in which the PIC16C71 waits to receive a command from a host through the RS-232. On reception of a byte (valid commands are 0x00, 0x01, 0x02 & 0x03), the received byte is treated as the PIC16C71's A/D channel number and the requested channel is selected, an A/D conversion is started and when the conversion is complete (in about 20 uS) the digital data (8 bits) are transmitted back to the host. A Microsoft® Windows® program running on an IBM PC/AT® was written to act as a host and collect the A/D data from PIC16C71 via an RS-232 port. The Windows program (DVM.EXE) runs as a background job and displays the A/D data in a small window (similar to the CLOCK program that comes with MS Windows). The windows program and the PIC16C71 together act like a data acquisition system or a digital volt meter (DVM). The block diagram of the system is shown in Figure 2. The input clock frequency is fixed at 4 MHz and RS-232 parameters are set to 1200 Baud, 8-bits, 1 Stop Bit and No Parity. The program during development stage was also tested at 1200, 2400, 4800 Baud Rates @ 4 MHz Input Clock and up to 19200 Baud @ 10 MHz input clock frequency (all tests were performed with No Parity, Even Parity and Odd Parity at 8 and 7 Data Bits).

FIGURE 5 - MS WINDOWS PROGRAM FETCHING A/D DATA FROM PIC16C71 VIA RS-232



© 1994 Microchip Technology Inc.

Source Code

Author: *Amar Palacherla*
Logic Products Division
Code Update: *Scott Fink - Sr. Applications Engineer*
Logic Products Division

The PIC16CXX source code along with the Microsoft® Windows™ DVM Program (executable running on an IBM PC/AT under MS Windows 3.1 or higher) is available on Microchip's BBS. The assembly code for PIC16CXX must be assembled using Microchip's Universal Assembler, MPASM. The code cannot be assembled using the older assemblers without significant modifications. It is suggested that user's who do not have the new assembler MPASM, must change to the new version.

The MS Windows Program (DVM.EXE) runs under MS Windows 3.1 or higher. The program does not have any menus and shows up as a small window displaying A/D Data and runs as a background job. There are a few command line options and are described below :

-Px : x is the comm port number (e.g. - P2 selects COM2). Default is COM1

-Cy : y is the number of A/D channels to display. Default is one channel (channel #1)

-Sz : z is a floating point number that represents the scaling factor (For example - S5.5 would display the data as 5.5*<8bit A/D>/256). The default value is 5.0 volts. -S0 will display the data in raw format without any scaling.

© 1994 Microchip Technology Inc.

Appendix A - RS232.H

```
        NOLIST
;*************************************************************************************
;                       RS-232 Header File
;  PIC16C6X/7X/8X
;*************************************************************************************

_ClkOut         equ     (_ClkIn >> 2)           ; Instruction Cycle Freq = CLKIN/4
;
_CyclesPerBit   set     (_ClkOut/_BaudRate)
_tempCompute    set     (_CyclesPerBit >> 8)
;
;*************************************************************************************
;       Auto Generation Of Prescaler & Rtcc Values
;   Computed during Assembly Time
;*************************************************************************************

;  At first set Default values for RtccPrescale & RtccPreLoad
;
RtccPrescale    set     0
RtccPreLoad     set     _CyclesPerBit
UsePrescale     set     FALSE

 if (_tempCompute >= 1)
  RtccPrescale set      0
  RtccPreLoad  set      (_CyclesPerBit >> 1)

    UsePrescale     set     TRUE
 endif
 if (_tempCompute >= 2)
  RtccPrescale set      1
  RtccPreLoad  set      (_CyclesPerBit >> 2)
 endif
 if (_tempCompute >= 4)
  RtccPrescale set      2
  RtccPreLoad  set      (_CyclesPerBit >> 3)
 endif

 if (_tempCompute >= 8)
  RtccPrescale set      3
  RtccPreLoad  set      (_CyclesPerBit >> 4)
 endif
```

© 1994 Microchip Technology Inc.

```
 if (_tempCompute >= 16)
  RtccPrescale set     4
  RtccPreLoad  set     (_CyclesPerBit >> 5)
 endif
 if (_tempCompute >= 32)
  RtccPrescale set     5
  RtccPreLoad  set     (_CyclesPerBit >> 6)
 endif
 if (_tempCompute >= 64)
  RtccPrescale set     6
  RtccPreLoad  set     (_CyclesPerBit >> 7)
 endif

 if   (_tempCompute >= 128)
  RtccPrescale set     7
  RtccPreLoad  set     (_CyclesPerBit >> 8)
 endif
;
    if( (RtccPrescale == 0) && (RtccPreLoad < 60))
      messg  "Warning : Baud Rate May Be Too High For This Input Clock"
    endif
;*****************************************************************************
; Compute RTCC & Presclaer Values For 1.5 Times the Baud Rate for Start Bit Detection
;*****************************************************************************
_SBitCycles    set     (_ClkOut/_BaudRate) + ((_ClkOut/4)/_BaudRate)
_tempCompute   set     (_SBitCycles >> 8)

_BIT1_INIT     set     08
SBitPrescale   set     0
SBitRtccLoad   set     _SBitCycles

 if (_tempCompute >= 1)
  SBitPrescale set     0
  SBitRtccLoad set     (_SBitCycles >> 1)
  _BIT1_INIT   set     0
 endif

 if (_tempCompute >= 2)
```


© 1994 Microchip Technology Inc.

```
 SBitPrescale set     1
 SBitRtccLoad set     (_SBitCycles >> 2)
endif

if (_tempCompute >= 4)
 SBitPrescale set     2
 SBitRtccLoad set     (_SBitCycles >> 3)
endif
if (_tempCompute >= 8)
 SBitPrescale set     3
 SBitRtccLoad set     (_SBitCycles >> 4)
endif


if (_tempCompute >= 16)
 SBitPrescale set     4
 SBitRtccLoad set     (_SBitCycles >> 5)
endif

if (_tempCompute >= 32)
 SBitPrescale set     5
 SBitRtccLoad set     (_SBitCycles >> 6)
endif

if (_tempCompute >= 64)
 SBitPrescale set     6
 SBitRtccLoad set     (_SBitCycles >> 7)
endif

if   (_tempCompute >= 128)
 SBitPrescale set     7
 SBitRtccLoad set     (_SBitCycles >> 8)
endif

;
;*****************************************************************************
;
#define _Cycle_Offset1 24                    ;account for interrupt latency, call time

LOAD_RTCC      MACRO   Mode, K, Prescale
   if(UsePrescale == 0 && Mode == 0)
```

© 1994 Microchip Technology Inc.

```
  movlw -K + _Cycle_Offset1
    else
  movlw -K + (_Cycle_Offset1 >> (Prescale+1))  ; Re Load RTCC init value + INT Latency Offset
    endif
  movwf _rtcc           ; Note that Prescaler is cleared when RTCC is written
        ENDM
;*************************************************************************************
LOAD_BITCOUNT  MACRO

  movlw _DataBits+1
  movwf BitCount
  movlw 1
  movwf ExtraBitCount
    if  _PARITY_ENABLE
  movlw 2
  movwf ExtraBitCount
    endif
        ENDM
;
;*********************************************************************************************
;             Pin Assignements
;*********************************************************************************************
#define RX_MASK 0x10          ; RX pin is connected to RA4, ie. bit 4
#define RX_Pin  _porta,4      ; RX Pin : RA4
#define RX      RxTemp,4

#define TX      _portb,7      ; TX Pin , RB7
#define _RTS    _portb,5      ; RTS Pin, RB5, Output signal
#define _CTS    _portb,6      ; CTS Pin, RB6, Input signal


#define _txmtProgress  SerialStatus,0
#define _txmtEnable    SerialStatus,1

#define _rcvProgress   SerialStatus,2
#define _rcvOver       SerialStatus,3
#define _ParityErr     SerialStatus,4
#define _FrameErr      SerialStatus,5

#define _parityBit     SerialStatus,7

;*********************************************************************************************
```


© 1994 Microchip Technology Inc.

```
_OPTION_SBIT   set     0x38          ; Increment on Ext Clock (falling edge), for START Bit Detect

 if UsePrescale
_OPTION_INIT   set     0x00          ; Prescaler is used depending on Input Clock & Baud Rate
 else
_OPTION_INIT   set     0x0F
 endif

      CBLOCK  0x0C
              TxReg                 ; Transmit Data Holding/Shift Reg
              RxReg                 ; Rcv Data Holding Reg
                      RxTemp
              SerialStatus          ; Txmt & Rev Status/Control Reg
              BitCount
              ExtraBitCount         ; Parity & Stop Bit Count
              SaveWReg              ; temp hold reg of WREG on INT
                      SaveStatus        ; temp hold reg of STATUS Reg on INT
 temp1, temp2
      ENDC
;*****************************************************************************************************
      LIST
```

© 1994 Microchip Technology Inc.

Appendix B - RS232.ASM

```
  TITLE           "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
  SUBTITLE        "Software Implementation : Interrupt Driven"

;*****************************************************************************************************
;                       Software Implementation Of RS232 Communications Using PIC16CXX
;                                           Half-Duplex
;
;  These routines are intended to be used with PIC16C6X/7X family. These routines can be
;  used with processors in the 16C6X/7X family which do not have on board Hardware Async
;  Serial Port.
;  MX..
;
;  Description :
;               Half Duplex RS-232 Mode Is implemented in Software.
;               Both Reception & Transmission are Interrupt driven
;               Only 1 peripheral (RTCC) used for both transmission & reception
;               RTCC is used for both timing generation (for bit transmission & bit polling)
;               and Start Bit Detection in reception mode.
;               This is explained in more detail in the Interrupt Subroutine.
;               Programmable Baud Rate (speed depnding on Input Clock Freq.), programmable
;               #of bits, Parity enable/disable, odd/even parity is implemented.
;                       Parity & Framing errors are detected on Reception
;
;                               RS-232 Parameters
;
;The RS-232 Parameters are defined as shown below:
;
;               _ClkIn          :       Input Clock Frequency of the processor
;                                       (NOTE : RC Clock Mode Is Not Suggested due to wide variations)
;               _BaudRate       :       Desired Baud Rate. Any valid value can be used.
;                                       The highest Baud Rate achievable depends on Input Clock Freq.
;                                       300 to 4800 Baud was tested using 4 Mhz Input Clock
;                                       300 to 19200 Baud was tested using 10 Mhz Input Clock
;                                       Higher rates can be obtained using higher Input Clock Frequencies.
;                                       Once the _BaudRate & _ClkIn are specified the program
;                                       automatically selectes all the appropiate timings
;               _DataBits       :       Can specify 1 to 8 Bits.
;               _StopBits       :       Limited to 1 Stop Bit. Must set it to 1.
;               _PARITY_ENABLE  :       Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
;                                       is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
;                                       description below
;               _ODD_PARITY     :       Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
;                                       EVEN Parity Scheme is used.
;                                       This Flag is ignored if _PARITY_ENABLE is set to FALSE.
;
```


© 1994 Microchip Technology Inc.

```
;  Usage :
;               An example is given in the main program on how to Receive & Transmit Data
;               In the example, the processor waits until a command is received. The command is interpreted
;               as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
;               is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
;
;                       The RS-232 Control/Status Reg's bits are explained below :
;
;       "SerialStatus"          : RS-232 Status/Control Register
;
;       Bit 0   :       _txmtProgress   (1 if transmission in progress, 0 if transmission is complete)
;                                       After a byte is transmitted by calling "PutChar" function, the
;                                       user's code can poll this bit to check if transmission is complete.
;                                       This bit is reset after the STOP bit has been transmitted
;       Bit 1   :       _txmtEnable     Set this bit to 1 on initialization to enable transmission.
;                                       This bit can be used to Abort a transmission while the transmitter
;                                       is in progress (i.e when _txmtProgress = 1)
;       Bit 2   :       _rcvProgress    Indicates that the receiver is in middle of reception.It is reset when
;                                       a byte is received.
;       Bit 3   :       _rcvOver        This bit indicates the completion of Reception of a Byte. The user's
;                                       code can poll this bit after calling "GetChar" function. Once "GetChar"
;                                       function is called, this bit is 1 and is set to 0 after reception of
;                                       a complete byte (parity bit if enabled & stop bit)
;       Bit 4   :       _ParityErr      A 1 indicates Parity Error on Reception (for both even & odd parity)
;       Bit 5   :       _FrameErr       A 1 indicates Framing Error On Reception
;
;       Bit 6   :       _unused_        Unimplemented Bit
;
;       Bit 7   :       _parityBit      The 9 th bit of transmission or reception (status of PARITY bit
;                                       if parity is enabled)
;
;       To Transmit A Byte Of Data :
;                       1) Make sure _txmtProgress & _rcvOver bits are cleared
;                       2) Load TxReg with data to be transmitted
;                       3) CALL  PutChar function
;
;       To Receive A Byte Of Data :
;                       1) Make sure _txmtProgress & _rcvOver bits are cleared
;                       2) CALL GetChar function
;                       3) The received Byte is in TxReg after _rcvOver bit is cleared
;
;
; Rev 2, May 17,1994 Scott Fink
;       Corrected 7 bit and parity operation, corrected stop bit generation, corrected
;       receive prescaler settings.  Protected against inadvertant WDT reset.
;*************************************************************************************************

        Processor       16C71
```

© 1994 Microchip Technology Inc.

```
        Radix   DEC
        EXPAND

        include         "16Cxx.h"
;*****************************************************************************************************
;                               Setup RS-232 Parameters
;*****************************************************************************************************

_ClkIn          equ     4000000         ; Input Clock Frequency is 4 Mhz
_BaudRate       set     1200            ; Baud Rate (bits per second) is 1200
_DataBits       set     8               ; 8 bit data, can be 1 to 8
_StopBits       set     1               ; 1 Stop Bit, 2 Stop Bits is not implemented

#define _PARITY_ENABLE  FALSE           ; NO Parity
#define _ODD_PARITY     FALSE           ; EVEN Parity, if Parity enabled
#define _USE_RTSCTS     FALSE           ; NO Hardware Handshaking is Used

        include         "rs232.h"

;*****************************************************************************************************
;
  ORG     _ResetVector
  goto    Start
;
  ORG     _IntVector
  goto    Interrupt
;*****************************************************************************************************
;                               Table Of ADCON0 Reg
; Inputs : WREG (valid values are 0 thru 3)
; Returns In WREG, ADCON0 Value, selecting the desired Channel
;
;  Program Memory :     6 locations
;  Cycles         :     5
;
;*****************************************************************************************************

GetADCon0:
  andlw   0x03                  ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
  addwf   _pcl
  retlw   (0xC1 | (0 << 3))     ; channel 0
  retlw   (0xC1 | (1 << 3))     ; channel 1
  retlw   (0xC1 | (2 << 3))     ; channel 2
GetADCon0_End:
  retlw   (0xC1 | (3 << 3))     ; channel 3
```


© 1994 Microchip Technology Inc.

```
   if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
  MESSG   "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
   endif
;
;*****************************************************************************************************
;                                Initialize A/D Converter
;  <RA0:RA3>    Configure as Analog Inputs, VDD as Vref
;   A/D Clock Is Internal RC Clock
;   Select Channel 0
;
;  Program Memory :     6 locations
;  Cycles         :     7
;
;*****************************************************************************************************
InitAtoD:
  bsf     _rp0
  clrf    _adcon1
  bcf     _rp0
  movlw   0xC1
  movwf   _adcon0
  return
;
;*****************************************************************************************************
;                                     Main Program Loop
;
;  After appropriate initilization, The main program wait for a command from RS-232
;  The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
;  After a command is received, the appropriate A/D Channel is seleted and when conversion is
;  completed the A/D Data is transmitted back to the Host. The controller now waits for a new
;  command.
;*****************************************************************************************************

Start:
  call    InitSerialPort
;
WaitForNextSel:
   if _USE_RTSCTS
  bcf     _rp0
  bcf     _RTS                    ; ready to accept data from host
   endif
  call    GetChar                 ; wait for a byte reception
  btfsc   _rcvOver                ; _rcvOver Gets Cleared when a Byte Is Received (in RxReg)
  goto    $-1                     ; USER can perform other jobs here, can poll _rcvOver bit
;
; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
;
  bcf     _rp0                    ; make sure to select Page 0
```


© 1994 Microchip Technology Inc.

```
  movf    RxReg,w                 ; WREG = Commanded Channel # (0 thru 3)
  call    GetADCon0               ; Get ADCON0 Reg Constant from Table Lookup
  movwf   _adcon0                 ; Load ADCON0 reg, selecting the desired channel
  nop
;
  bsf     _go                     ; start conversion
  btfsc   _done
  goto    $-1                     ; Loop Until A/D Conversion Done

  movf    _adres,w
  movwf   TxReg
   if _USE_RTSCTS
  bsf     _RTS                    ; Half duplex mode, transmission mode, ask host not to send data
  btfsc   _CTS                    ; Check CTS signal if host ready to accept data
  goto    $-1
   endif
  call    PutChar
  btfsc   _txmtProgress
  goto    $-1                     ; Loop Until Transmission Over, User Can Perform Other Jobs
;
  goto    WaitForNextSel          ; wait for next selection (command from Serial Port)
;
;*****************************************************************************************************
;                                RS-232 Routines
;;****************************************************************************************************
;                                Interrupt Service Routine
;
; Only RTCC Inturrupt Is used. RTCC Inturrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
; Transmit Modules.
;       Transmission :
;                       RTCC is setup for Internal Clock increments and interrupt is generated when
;                       RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
;                       desired BAUD RATE.
;       Reception :
;                       When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
;                       and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC
;                       rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
;                       bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
;                       with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
;
;*****************************************************************************************************

Interrupt:
  btfss   _rtif
  retfie                          ; other interrupt, simply return & enable GIE
```


© 1994 Microchip Technology Inc.

```
;
; Save Status On INT : WREG & STATUS Regs
;
  movwf   SaveWReg
  swapf   _status,w              ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
  movwf   SaveStatus
;
  btfsc   _txmtProgress
  goto    _TxmtNextBit           ; Txmt Next Bit
  btfsc   _rcvProgress
  goto    _RcvNextBit            ; Receive Next Bit
  goto    _SBitDetected          ; Must be start Bit
;
RestoreIntStatus:
  swapf   SaveStatus,w
  movwf   _status                ; restore STATUS Reg
  swapf   SaveWReg               ; save WREG
  swapf   SaveWReg,w             ; restore WREG
  bcf     _rtif
  retfie
;
;*****************************************************************************************************
;
;
; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
; Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception
;
;  Program Memory :     9 locations
;  Cycles         :     10
;*****************************************************************************************************

InitSerialPort:
  clrf    SerialStatus
;
  bcf     _rp0                         ; select Page 0 for Port Access
  bsf     TX                           ; make sure TX Pin is high on powerup, use RB Port Pullup
  bsf     _rp0                         ; Select Page 1 for TrisB access
  bcf     TX                           ; set TX Pin As Output Pin, by modifying TRIS
  if _USE_RTSCTS
  bcf     _RTS                         ; RTS is output signal, controlled by PIC16Cxx
  bsf     _CTS                         ; CTS is Input signal, controlled by the host
  endif
  bsf     RX_Pin                       ; set RX Pin As Input for reception
  return
;
;*****************************************************************************************************
```

© 1994 Microchip Technology Inc.

```
include "txmtr.asm"  ; The Transmit routines are in file "txmtr.asm"
include "rcvr.asm"   ; The Receiver Routines are in File "rcvr.asm"

;*****************************************************************************

  END
```


© 1994 Microchip Technology Inc.

Appendix C - RCVR.ASM

```
;*******************************************************************************
;             GetChar Function
;  Receives a Byte Of Data
;       When reception is complete, _rcvOver Bit is cleared
;       The received data is in RxReg
;
;  Program Memory :    15 locations (17 locations if PARITY is used)
;  Cycles         :    16 (18 if PARITY is USED)
;
;*******************************************************************************
GetChar:
  bcf   _rp0
  bsf   _rcvOver                    ; Enable Reception, this bit gets reset on Byte Rcv Complete
  LOAD_BITCOUNT
        clrf   RxReg
  bcf   _FrameErr
        bcf    _ParityErr           ; Init Parity & Framing Errors
  clrf  _rtcc
  clrwdt
  bsf   _rp0
  movlw 07h
  movwf _option
  bcf   _rp0
  clrf  _rtcc
  bsf   _rp0
  movlw 0Fh
  movwf _option
  clrwdt
        movlw  _OPTION_SBIT         ; Inc On Ext Clk Falling Edge
  movwf _option                     ; Set Option Reg Located In Page 1
  bcf   _rp0                        ; make sure to select Page 0
  movlw 0xFF
  movwf _rtcc                       ; A Start Bit will roll over RTCC & Gen INT
  bcf   _rtif
  bsf   _rtie                       ; Enable RTCC Interrupt
  retfie                            ; Enable Global Interrupt
;
;*******************************************************************************
;             Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :    14 locations
;  Cycles         :    12 (worst case)
;
;*******************************************************************************
_SBitDetected:
```

© 1994 Microchip Technology Inc.

```
  bcf   _rp0
  btfsc RX_Pin                          ; Make sure Start Bit Interrupt is not a Glitch
  goto  _FalseStartBit                  ; False Start Bit
  bsf   _rcvProgress
  clrf  _rtcc
  clrwdt
  bsf   _rp0
  movlw 07h
  movwf _option
  bcf   _rp0
  clrf  _rtcc
  bsf   _rp0
  movlw 0Fh
  movwf _option
  clrwdt
  movlw (_BIT1_INIT | SBitPrescale)     ; Switch Back to INT Clock
  movwf _option                         ; Set Option Reg Located In Page 1
  bcf   _rp0                            ; make sure to select Page 0
  LOAD_RTCC  1,(SBitRtccLoad), SBitPrescale
  goto  RestoreIntStatus
;
_FalseStartBit:
  movlw 0xFF
        movwf  _rtcc                    ; reload RTCC with 0xFF for start bit detection
  goto  RestoreIntStatus
;
;*****************************************************************************************
;             Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :    28 locations ( 43 locations with PARITY enabled)
;  Cycles         :     24 Worst Case
;
;*****************************************************************************************
_RcvNextBit:
  clrwdt
  bsf   _rp0
  movlw 07h
  movwf _option
  bcf   _rp0
  clrf  _rtcc
  clrwdt
  bsf   _rp0
  movlw 07h
  movwf _option
  bcf   _rp0
  clrf  _rtcc
  bsf   _rp0
```


© 1994 Microchip Technology Inc.

```
  movlw 0Fh
  movwf _option
  clrwdt
  movlw (_OPTION_INIT | RtccPrescale)        ; Switch Back to INT Clock
  movwf _option                              ; Set Option Reg Located In Page 1
;
  bcf   _rp0
  movf  _porta,w                             ; read RX pin immediately into WREG
  movwf RxTemp
  LOAD_RTCC  0,RtccPreLoad, RtccPrescale     ; Macro to reload RTCC
  movf  _porta,w
  xorwf RxTemp,w
  andlw RX_MASK                              ; mask for only RX PIN (RA4)
  btfsc _z
  goto  _PinSampled                          ; both samples are same state
_SampleAgain:
  movf  _porta,w
  movwf RxTemp                               ; 2 out of 3 majority sampling done
_PinSampled:
  movf  BitCount,1
  btfsc _z
  goto  _RcvP_Or_S
;
  decfsz       BitCount
  goto  _NextRcvBit
;
_RcvP_Or_S:
   if _PARITY_ENABLE
  decfsz       ExtraBitCount
       goto    _RcvParity
   endif
;
_RcvStopBit:
  btfss RX
  bsf   _FrameErr                            ; may be framing Error or Glitch
  bcf   _rtie                                ; disable further interrupts
  bcf   _rcvProgress
  bcf   _rcvOver                             ; Byte Received, Can RCV/TXMT an other Byte
  if _PARITY_ENABLE
  movf  RxReg,w
  call  GenParity                            ; Generate Parity, for Parity check
  movlw 0
  btfsc _parityBit
  movlw 0x10                                 ; to mask off Received Parity Bit in _ParityErr
  xorwf SerialStatus                         ; _ParityErr bit is set accordingly
  endif
  if _DataBits == 7
  rrf   RxReg,1
```

© 1994 Microchip Technology Inc.

```
    bcf   RxReg,7
    endif
    goto  RestoreIntStatus
;
_NextRcvBit:
    bcf   _carry
    btfsc RX                ; prepare bit for shift
    bsf   _carry
    rrf   RxReg             ; shift in received data
    goto  RestoreIntStatus
;
    if _PARITY_ENABLE
_RcvParity:
    bcf   _ParityErr        ; Temporarily store PARITY Bit in _ParityErr
    btfsc RX                ; Sample again to avoid any glitches
    bsf   _ParityErr
    goto  RestoreIntStatus
    endif
;
;*****************************************************************************
```


© 1994 Microchip Technology Inc.

Appendix D - TXMTR.ASM

```
;*****************************************************************************************************
;                       PutChar Function
;
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
;  Program Memory :    6 locations (10 locations if PARITY is Used)
;  Cycles         :    8 (13 if PARITY is Used)
;
;*****************************************************************************************************
PutChar:
  bsf   _txmtEnable                     ; enable transmission
  bsf   _txmtProgress
  LOAD_BITCOUNT                         ; Macro to load bit count
  decf  BitCount,1
  if _DataBits == 7
  bsf   TxReg,7
  endif
;
   if _PARITY_ENABLE
  movf  TxReg,W
  call  GenParity                       ; If Parity is used, then Generate Parity Bit
   endif
;
  call  _TxmtStartBit
  bsf   _rtie                           ; Enable RTCC Overflow INT
  retfie                                ; return with _GIE Bit Set
;
;*****************************************************************************************************
;             Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :   30 locations  (38 locations if PARITY is used)
;  Cycles         :   15 Worst Case
;
;*****************************************************************************************************

_TxmtNextBit:
  bcf   _rp0
  LOAD_RTCC  0,RtccPreLoad, RtccPrescale     ; Macro to reload RTCC
;
  movf  BitCount                        ;done with data xmission?
  btfsc _z
  goto  _ParityOrStop                   ;yes, do parity or stop bit
```


© 1994 Microchip Technology Inc.

```
;
  decf  BitCount
  goto  _NextTxmtBit                    ;no, send another
;
_ParityOrStop:
    if _PARITY_ENABLE
  btfsc ExtraBitCount,1                 ;ready for parity bit?
        goto   _SendParity
    endif
  movf  ExtraBitCount,1                 ;check if sending stop bit
  btfsc _z
  goto  DoneTxmt
  decf  ExtraBitCount,1
;
_StopBit:
  bsf   TX                              ; STOP Bit is High
  goto  RestoreIntStatus
  goto  DoneTxmt
;
_NextTxmtBit:
  bsf   _carry
  rrf   TxReg
  btfss _carry
        bcf    TX
  btfsc _carry
  bsf   TX
;
  btfss _txmtEnable
  bsf   _rtie                           ; disable further interrupts, Transmission Aborted
;
  goto  RestoreIntStatus
;
  if _PARITY_ENABLE
_SendParity:
  decf  ExtraBitCount,1                 ;subtract parity from count
  btfss _parityBit
  bcf   TX
  btfsc _parityBit
        bsf    TX
  goto  RestoreIntStatus
  endif
DoneTxmt
  bsf   TX                              ;STOP Bit is High
  bcf   _rtie                           ;disable further interrupts
  bcf   _txmtProgress                   ;indicates end of xmission
  goto  RestoreIntStatus
;
```


© 1994 Microchip Technology Inc.

```
;*****************************************************************************************************
;              Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :    9 locations
;  Cycles         :    10
;
;*****************************************************************************************************
_TxmtStartBit:
  bcf   _rp0
  clrf  _rtcc
  clrwdt
  bsf   _rp0
  movlw 07h
  movwf _option
  bcf   _rp0
  clrf  _rtcc
  bsf   _rp0
  movlw 0Fh
  movwf _option
  clrwdt
  movlw (_OPTION_INIT | RtccPrescale)
  movwf _option                         ; Set Option Reg Located In Page 1
  bcf   _rp0                            ; make sure to select Page 0
  bcf   TX                              ; Send Start Bit
  movlw -RtccPreLoad                    ; Prepare for Timing Interrupt
  movwf _rtcc
  bcf   _rtif
  return

;*****************************************************************************************************
;                    Generate Parity for the Value in WREG
;
;  The parity bit is set in _parityBit (SerialStatus,7)
;  Common Routine For Both Transmission & Reception
;
;  Program Memory :    16 locations
;  Cycles         :    72
;
;*****************************************************************************************************
  if _PARITY_ENABLE

GenParity:
  movwf temp2                           ;save data
  movf  BitCount,w                      ;save bitcount
  movwf temp1
Parityloop
      rrf   temp2
```


© 1994 Microchip Technology Inc.

```
  btfss _carry                          ;put data in carry bit
  goto  NotOne
  xorlw 00h                             ;parity calculated by XORing all data bits
  goto  OneDone
NotOne
  xorlw 01h
OneDone
  decfsz        temp1
  goto  Parityloop                      ;decrement count
  movwf temp1
; Parity bit is in Bit 0 of temp1
;
   if _ODD_PARITY
  bsf   _parityBit
  btfsc temp1,0
        bcf     _parityBit
    else
  bcf   _parityBit
  btfsc temp1,0
        bsf     _parityBit
   endif

  return
  endif
;*****************************************************************************************************
```


© 1994 Microchip Technology Inc.

Appendix E - RS232.LST

```
MPASM 01.00.02 Alpha  \PICMASTR\CU   5-20-1994  9:13:56                    PAGE  1
RS232 Communications : Half Duplex : PIC16C6x/7x/8x
Software Implementation : Interrupt Driven
LOC  OBJECT CODE     LINE SOURCE TEXT

                      0001         TITLE           "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
                      0002         SUBTITLE        "Software Implementation : Interrupt Driven"
                      0003
                      0004 ;*****************************************************************************************************
                      0005 ;                       Software Implementation Of RS232 Communications Using PIC16CXX
                      0006 ;                                          Half-Duplex
                      0007 ;
                      0008 ;  These routines are intended to be used with PIC16C6X/7X family. These routines can be
                      0009 ;  used with processors in the 16C6X/7X family which do not have on board Hardware Async
                      0010 ;  Serial Port.
                      0011 ;  MX..
                      0012 ;
                      0013 ;  Description :
                      0014 ;               Half Duplex RS-232 Mode Is implemented in Software.
                      0015 ;               Both Reception & Transmission are Interrupt driven
                      0016 ;               Only 1 peripheral (RTCC) used for both transmission & reception
                      0017 ;               RTCC is used for both timing generation (for bit transmission & bit polling)
                      0018 ;               and Start Bit Detection in reception mode.
                      0019 ;               This is explained in more detail in the Interrupt Subroutine.
                      0020 ;               Programmable Baud Rate (speed depnding on Input Clock Freq.), programmable
                      0021 ;               #of bits, Parity enable/disable, odd/even parity is implemented.
                      0022 ;                       Parity & Framing errors are detected on Reception
                      0023 ;
                      0024 ;                                  RS-232 Parameters
                      0025 ;
                      0026 ;The RS-232 Parameters are defined as shown below:
                      0027 ;
                      0028 ;               _ClkIn          :       Input Clock Frequency of the processor
                      0029 ;                                       (NOTE : RC Clock Mode Is Not Suggested due to wide variations)
                      0030 ;               _BaudRate       :       Desired Baud Rate. Any valid value can be used.
                      0031 ;                                       The highest Baud Rate achievable depends on Input Clock Freq.
                      0032 ;                                       300 to 4800 Baud was tested using 4 Mhz Input Clock
                      0033 ;                                       300 to 19200 Baud was tested using 10 Mhz Input Clock
                      0034 ;                                       Higher rates can be obtained using higher Input Clock Frequencies.
                      0035 ;                                       Once the _BaudRate & _ClkIn are specified the program
                      0036 ;                                       automatically selectes all the appropiate timings
                      0037 ;               _DataBits       :       Can specify 1 to 8 Bits.
                      0038 ;               _StopBits       :       Limited to 1 Stop Bit. Must set it to 1.
                      0039 ;               _PARITY_ENABLE  :       Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
```

© 1994 Microchip Technology Inc.

```
0040 ;                                          is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
0041 ;                                          description below
0042 ;             _ODD_PARITY     :           Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
0043 ;                                          EVEN Parity Scheme is used.
0044 ;                                          This Flag is ignored if _PARITY_ENABLE is set to FALSE.
0045 ;
0046 ;
0047 ; Usage :
0048 ;             An example is given in the main program on how to Receive & Transmit Data
0049 ;             In the example, the processor waits until a command is received. The command is interpreted
0050 ;             as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
0051 ;             is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
0052 ;
0053 ;                     The RS-232 Control/Status Reg's bits are explained below :
0054 ;
0055 ;     "SerialStatus"          : RS-232 Status/Control Register
0056 ;
0057 ;     Bit 0   :       _txmtProgress   (1 if transmission in progress, 0 if transmission is complete)
0058 ;                                     After a byte is transmitted by calling "PutChar" function, the
0059 ;                                     user's code can poll this bit to check if transmission is complete.
0060 ;                                     This bit is reset after the STOP bit has been transmitted
0061 ;     Bit 1   :       _txmtEnable     Set this bit to 1 on initialization to enable transmission.
0062 ;                                     This bit can be used to Abort a transmission while the transmitter
0063 ;                                     is in progress (i.e when _txmtProgress = 1)
0064 ;     Bit 2   :       _rcvProgress    Indicates that the receiver is in middle of reception.It is reset when
0065 ;                                     a byte is received.
0066 ;     Bit 3   :       _rcvOver        This bit indicates the completion of Reception of a Byte. The user's
0067 ;                                     code can poll this bit after calling "GetChar" function. Once "GetChar"
0068 ;                                     function is called, this bit is 1 and is set to 0 after reception of
0069 ;                                     a complete byte (parity bit if enabled & stop bit)
0070 ;     Bit 4   :       _ParityErr      A 1 indicates Parity Error on Reception (for both even & odd parity)
0071 ;     Bit 5   :       _FrameErr       A 1 indicates Framing Error On Reception
0072 ;
0073 ;     Bit 6   :       _unused_        Unimplemented Bit
0074 ;
0075 ;     Bit 7   :       _parityBit      The 9 th bit of transmission or reception (status of PARITY bit
0076 ;                                     if parity is enabled)
0077 ;
0078 ;     To Transmit A Byte Of Data :
0079 ;                     1) Make sure _txmtProgress & _rcvOver bits are cleared
0080 ;                     2) Load TxReg with data to be transmitted
0081 ;                     3) CALL  PutChar function
0082 ;
0083 ;     To Receive A Byte Of Data :
0084 ;                     1) Make sure _txmtProgress & _rcvOver bits are cleared
0085 ;                     2) CALL GetChar function
0086 ;                     3) The received Byte is in TxReg after _rcvOver bit is cleared
0087 ;
```


© 1994 Microchip Technology Inc.

```
                0088 ;
                0089 ;   Rev 2, May 17,1994 Scott Fink
                0090 ;          Corrected 7 bit and parity operation, corrected stop bit generation, corrected
                0091 ;          receive prescaler settings.  Protected against inadvertant WDT reset.
                0092 ;*****************************************************************************************************
                0093
                0094            Processor       16C71
                0095            Radix   DEC
                0096            EXPAND
                0097
                0098            include         "16Cxx.h"
                0179
                0180
                0181
                0098
                0099
                0100 ;*****************************************************************************************************
                0101 ;                                 Setup RS-232 Parameters
                0102 ;*****************************************************************************************************
                0103
003D 0900       0104 _ClkIn          equ     4000000         ; Input Clock Frequency is 4 Mhz
04B0            0105 _BaudRate       set     1200            ; Baud Rate (bits per second) is 1200
0008            0106 _DataBits       set     8               ; 8 bit data, can be 1 to 8
0001            0107 _StopBits       set     1               ; 1 Stop Bit, 2 Stop Bits is not implemented
                0108
0046            0109  #define _PARITY_ENABLE  FALSE           ; NO Parity
0047            0110  #define _ODD_PARITY     FALSE           ; EVEN Parity, if Parity enabled
0048            0111  #define _USE_RTSCTS     FALSE           ; NO Hardware Handshaking is Used
                0112
                0113            include         "rs232.h"
                0001
                0113
                0114
                0115 ;*****************************************************************************************************
                0116 ;
                0117
                0118        ORG     _ResetVector
0000 2811       0119        goto    Start
                0120 ;
                0121
                0122        ORG     _IntVector
0004 2823       0123        goto    Interrupt
                0124 ;
                0125 ;*****************************************************************************************************
                0126 ;                                 Table Of ADCON0 Reg
                0127 ; Inputs : WREG (valid values are 0 thru 3)
                0128 ; Returns In WREG, ADCON0 Value, selecting the desired Channel
                0129 ;
```

© 1994 Microchip Technology Inc.

```
                        0130 ;  Program Memory :     6 locations
                        0131 ;  Cycles         :     5
                        0132 ;
                        0133 ;*****************************************************************************************************
                        0134
                        0135 GetADCon0:
0005 3903               0136         andlw   0x03                    ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
0006 0782               0137         addwf   _pcl
0007 34C1               0138         retlw   (0xC1 | (0 << 3))       ; channel 0
0008 34C9               0139         retlw   (0xC1 | (1 << 3))       ; channel 1
0009 34D1               0140         retlw   (0xC1 | (2 << 3))       ; channel 2
                        0141 GetADCon0_End:
000A 34D9               0142         retlw   (0xC1 | (3 << 3))       ; channel 3
                        0143
                        0144    if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
                        0145         MESSG   "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
                        0146    endif
                        0147 ;
                        0148 ;*****************************************************************************************************
                        0149 ;                              Initialize A/D Converter
                        0150 ;  <RA0:RA3>    Configure as Analog Inputs, VDD as Vref
                        0151 ;   A/D Clock Is Internal RC Clock
                        0152 ;   Select Channel 0
                        0153 ;
                        0154 ;  Program Memory :     6 locations
                        0155 ;  Cycles         :     7
                        0156 ;
                        0157 ;*****************************************************************************************************
                        0158 InitAtoD:
000B 1683               0159         bsf     _rp0
000C 0188               0160         clrf    _adcon1
000D 1283               0161         bcf     _rp0
000E 30C1               0162         movlw   0xC1
000F 0088               0163         movwf   _adcon0
0010 0008               0164         return
                        0165 ;
                        0166 ;*****************************************************************************************************
                        0167 ;                                   Main Program Loop
                        0168 ;
                        0169 ;  After appropriate initilization, The main program wait for a command from RS-232
                        0170 ;  The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
                        0171 ;  After a command is received, the appropriate A/D Channel is seleted and when conversion is
                        0172 ;  completed the A/D Data is transmitted back to the Host. The controller now waits for a new
                        0173 ;  command.
                        0174 ;*****************************************************************************************************
                        0175
                        0176 Start:
0011 2033               0177         call    InitSerialPort
```


© 1994 Microchip Technology Inc.

```
                    0178 ;
                    0179 WaitForNextSel:
                    0180   if _USE_RTSCTS
                    0181       bcf     _rp0
                    0182       bcf     _RTS                ; ready to accept data from host
                    0183   endif
0012 2074           0184       call    GetChar             ; wait for a byte reception
0013 198F           0185       btfsc   _rcvOver            ; _rcvOver Gets Cleared when a Byte Is Received (in RxReg)
0014 2813           0186       goto    $-1                 ; USER can perform other jobs here, can poll _rcvOver bit
                    0187 ;
                    0188 ; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
                    0189 ;
0015 1283           0190       bcf     _rp0                ; make sure to select Page 0
0016 080D           0191       movf    RxReg,w             ; WREG = Commanded Channel # (0 thru 3)
0017 2005           0192       call    GetADCon0           ; Get ADCON0 Reg Constant from Table Lookup
0018 0088           0193       movwf   _adcon0             ; Load ADCON0 reg, selecting the desired channel
0019 0000           0194       nop
                    0195 ;
001A 1508           0196       bsf     _go                 ; start conversion
001B 1908           0197       btfsc   _done
001C 281B           0198       goto    $-1                 ; Loop Until A/D Conversion Done
                    0199
001D 0809           0200       movf    _adres,w
001E 008C           0201       movwf   TxReg
                    0202   if _USE_RTSCTS
                    0203       bsf     _RTS                ; Half duplex mode, transmission mode, ask host not to send data
                    0204       btfsc   _CTS                ; Check CTS signal if host ready to accept data
                    0205       goto    $-1
                    0206   endif
ww001F 203A         0207       call    PutChar
0020 180F           0208       btfsc   _txmtProgress
0021 2820           0209       goto    $-1                 ; Loop Until Transmission Over, User Can Perform Other Jobs
                    0210
                    0211
                    0212 ;
0022 2812           0213       goto    WaitForNextSel      ; wait for next selection (command from Serial Port)
                    0214 ;
                    0215 ;*************************************************************************************************
                    0216 ;                                 RS-232 Routines
                    0217 ;;************************************************************************************************
                    0218 ;                                 Interrupt Service Routine
                    0219 ;
                    0220 ; Only RTCC Inturrupt Is used. RTCC Inturrupt is used as timing for Serial Port Receive & Transmit
                    0221 ; Since RS-232 is implemented only as a Half Duplex System, The RTCC is shared by both Receive &
                    0222 ; Transmit Modules.
                    0223 ;       Transmission :
                    0224 ;                   RTCC is setup for Internal Clock increments and interrupt is generated when
```

© 1994 Microchip Technology Inc.

```
                        0225 ;                  RTCC overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
                        0226 ;                  desired BAUD RATE.
                        0227 ;       Reception :
                        0228 ;                  When put in receive mode, RTCC is setup for external clock mode (FALLING EDGE)
                        0229 ;                  and preloaded with 0xFF. When a Falling Edge is detected on RTCC Pin, RTCC
                        0230 ;                  rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
                        0231 ;                  bit is detected, RTCC is changed to INTERNAL CLOCK mode and RTCC is preloaded
                        0232 ;                  with a certain value for regular timing interrupts to Poll RTCC Pin (i.e RX pin).
                        0233 ;
                        0234 ;*****************************************************************************************************
                        0235
                        0236 Interrupt:
0023 1D0B               0237         btfss   _rtif
0024 0009               0238         retfie                      ; other interrupt, simply return & enable GIE
                        0239 ;
                        0240 ; Save Status On INT : WREG & STATUS Regs
                        0241 ;
0025 0092               0242         movwf   SaveWReg
0026 0E03               0243         swapf   _status,w           ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0027 0093               0244         movwf   SaveStatus
                        0245 ;
0028 180F               0246         btfsc   _txmtProgress
0029 2844               0247         goto    _TxmtNextBit        ; Txmt Next Bit
002A 190F               0248         btfsc   _rcvProgress
002B 28A8               0249         goto    _RcvNextBit         ; Receive Next Bit
002C 2890               0250         goto    _SBitDetected       ; Must be start Bit
                        0251 ;
                        0252 RestoreIntStatus:
002D 0E13               0253         swapf   SaveStatus,w
002E 0083               0254         movwf   _status             ; restore STATUS Reg
002F 0E92               0255         swapf   SaveWReg            ; save WREG
0030 0E12               0256         swapf   SaveWReg,w          ; restore WREG
0031 110B               0257         bcf     _rtif
0032 0009               0258         retfie
                        0259 ;
                        0260 ;*****************************************************************************************************
                        0261 ;
                        0262 ;
                        0263 ;
                        0264 ; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
                        0265 ; Configure, RX_Pin (RTCC pin) as Input, which is used to poll data on reception
                        0266 ;
                        0267 ;  Program Memory :    9 locations
                        0268 ;  Cycles          :    10
                        0269 ;*****************************************************************************************************
                        0270
                        0271 InitSerialPort:
```


© 1994 Microchip Technology Inc.

```
0033 018F          0272          clrf    SerialStatus
                   0273 ;
0034 1283          0274          bcf     _rp0                    ; select Page 0 for Port Access
0035 1786          0275          bsf     TX                      ; make sure TX Pin is high on powerup, use RB Port Pullup
0036 1683          0276          bsf     _rp0                    ; Select Page 1 for TrisB access
0037 1386          0277          bcf     TX                      ; set TX Pin As Output Pin, by modifying TRIS
                   0278  if _USE_RTSCTS
                   0279          bcf     _RTS                    ; RTS is output signal, controlled by PIC16Cxx
                   0280          bsf     _CTS                    ; CTS is Input signal, controlled by the host
                   0281  endif
0038 1605          0282          bsf     RX_Pin                  ; set RX Pin As Input for reception
0039 0008          0283          return
                   0284 ;
                   0285 ;*****************************************************************************************************
                   0286
                   0287    include "txmtr.asm"    ; The Transmit routines are in file "txmtr.asm"
                   0001 ;*****************************************************************************************************
                   0002 ;                          PutChar Function
                   0003 ;
                   0004 ;   Function to transmit A Byte Of Data
                   0005 ;   Before calling this routine, load the Byte to be transmitted into TxReg
                   0006 ;   Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
                   0007 ;   calling this routine
                   0008 ;
                   0009 ;  Program Memory :        6 locations (10 locations if PARITY is Used)
                   0010 ;  Cycles       :     8 (13 if PARITY is Used)
                   0011 ;
                   0012 ;*****************************************************************************************************
                   0013 PutChar:
003A 148F          0014          bsf     _txmtEnable             ; enable transmission
003B 140F          0015          bsf     _txmtProgress
                   0016    LOAD_BITCOUNT                         ; Macro to load bit count
003C 3009            M           movlw   _DataBits+1
003D 0090            M           movwf   BitCount
003E 3001            M           movlw   1
003F 0091            M           movwf   ExtraBitCount
                     M    if  _PARITY_ENABLE
                     M           movlw   2
                     M           movwf   ExtraBitCount
                     M    endif
0040 0390          0017          decf    BitCount,1
                   0018  if _DataBits == 7
                   0019          bsf     TxReg,7
                   0020  endif
                   0021 ;
                   0022   if _PARITY_ENABLE
                   0023          movf    TxReg,W
                   0024          call    GenParity               ; If Parity is used, then Generate Parity Bit
```


© 1994 Microchip Technology Inc.

```
                    0025    endif
                    0026 ;
0041 2060           0027        call    _TxmtStartBit
0042 168B           0028        bsf     _rtie                       ; Enable RTCC Overflow INT
0043 0009           0029        retfie                              ; return with _GIE Bit Set
                    0030 ;
                    0031 ;*************************************************************************************************
                    0032 ;                 Internal Subroutine
                    0033 ; entered from Interrupt Service Routine when Start Bit Is detected.
                    0034 ;
                    0035 ;  Program Memory :    30 locations  (38 locations if PARITY is used)
                    0036 ;  Cycles    :   15 Worst Case
                    0037 ;
                    0038 ;*************************************************************************************************
                    0039
                    0040 _TxmtNextBit:
0044 1283           0041        bcf     _rp0
                    0042    LOAD_RTCC  0,RtccPreLoad, RtccPrescale      ; Macro to reload RTCC
                     M      if(UsePrescale == 0 && 0 == 0)
                     M          movlw   -RtccPreLoad + _Cycle_Offset1
                     M      else
0045 3036            M          movlw   -RtccPreLoad + (_Cycle_Offset1 >> (RtccPrescale+1))  ; Re Load RTCC init value + INT La
                     M      endif
0046 0081            M          movwf   _rtcc                       ; Note that Prescaler is cleared when RTCC is written
                    0043 ;
0047 0890           0044        movf    BitCount                    ;done with data xmission?
0048 1903           0045        btfsc   _z
0049 284C           0046        goto    _ParityOrStop               ;yes, do parity or stop bit
                    0047 ;
004A 0390           0048        decf    BitCount
004B 2853           0049        goto    _NextTxmtBit                ;no, send another
                    0050 ;
                    0051 _ParityOrStop:
                    0052    if _PARITY_ENABLE
                    0053        btfsc   ExtraBitCount,1             ;ready for parity bit?
                    0054        goto    _SendParity
                    0055    endif
004C 0891           0056        movf    ExtraBitCount,1             ;check if sending stop bit
004D 1903           0057        btfsc   _z
004E 285C           0058        goto    DoneTxmt
004F 0391           0059        decf    ExtraBitCount,1
                    0060 ;
                    0061 _StopBit:
0050 1786           0062        bsf     TX                          ; STOP Bit is High
0051 282D           0063        goto    RestoreIntStatus
0052 285C           0064        goto    DoneTxmt
                    0065 ;
```


© 1994 Microchip Technology Inc.

```
                        0066 _NextTxmtBit:
0053 1403               0067         bsf     _carry
0054 0C8C               0068         rrf     TxReg
0055 1C03               0069         btfss   _carry
0056 1386               0070         bcf     TX
0057 1803               0071         btfsc   _carry
0058 1786               0072         bsf     TX
                        0073 ;
0059 1C8F               0074         btfss   _txmtEnable
005A 168B               0075         bsf     _rtie                   ; disable further interrupts, Transmission Aborted
                        0076 ;
005B 282D               0077         goto    RestoreIntStatus
                        0078 ;
                        0079   if _PARITY_ENABLE
                        0080 _SendParity:
                        0081         decf    ExtraBitCount,1         ;subtract parity from count
                        0082         btfss   _parityBit
                        0083         bcf     TX
                        0084         btfsc   _parityBit
                        0085         bsf     TX
                        0086         goto    RestoreIntStatus
                        0087   endif
                        0088
                        0089 DoneTxmt
005C 1786               0090         bsf     TX                      ;STOP Bit is High
005D 128B               0091         bcf     _rtie                   ;disable further interrupts
005E 100F               0092         bcf     _txmtProgress           ;indicates end of xmission
005F 282D               0093         goto    RestoreIntStatus
                        0094 ;*****************************************************************************************************
                        0095 ;*****************************************************************************************************
                        0096 ;                 Internal Subroutine
                        0097 ; entered from Interrupt Service Routine when Start Bit Is detected.
                        0098 ;
                        0099 ;  Program Memory :      9 locations
                        0100 ;  Cycles    :     10
                        0101 ;
                        0102 ;*****************************************************************************************************
                        0103 _TxmtStartBit:
0060 1283               0104         bcf     _rp0
0061 0181               0105         clrf    _rtcc
0062 0064               0106         clrwdt
0063 1683               0107         bsf     _rp0
0064 3007               0108         movlw   07h
0065 0081               0109         movwf   _option
0066 1283               0110         bcf     _rp0
0067 0181               0111         clrf    _rtcc
0068 1683               0112         bsf     _rp0
0069 300F               0113         movlw   0Fh
```


© 1994 Microchip Technology Inc.

```
006A 0081          0114          movwf   _option
006B 0064          0115          clrwdt
006C 3001          0116          movlw   (_OPTION_INIT | RtccPrescale)
006D 0081          0117          movwf   _option                 ; Set Option Reg Located In Page 1
006E 1283          0118          bcf     _rp0                    ; make sure to select Page 0
006F 1386          0119          bcf     TX                      ; Send Start Bit
0070 3030          0120          movlw   -RtccPreLoad            ; Prepare for Timing Interrupt
0071 0081          0121          movwf   _rtcc
0072 110B          0122          bcf     _rtif
0073 0008          0123          return
                   0124
                   0125 ;*****************************************************************************************************
                   0126 ;                          Generate Parity for the Value in WREG
                   0127 ;
                   0128 ;  The parity bit is set in _parityBit (SerialStatus,7)
                   0129 ;  Common Routine For Both Transmission & Reception
                   0130 ;
                   0131 ;  Program Memory :     16 locations
                   0132 ;  Cycles     :     72
                   0133 ;
                   0134 ;*****************************************************************************************************
                   0135   if _PARITY_ENABLE
                   0136
                   0137 GenParity:
                   0138          movwf   temp2                   ;save data
                   0139          movf    BitCount,w              ;save bitcount
                   0140          movwf   temp1
                   0141 Parityloop
                   0142          rrf     temp2
                   0143          btfss   _carry                  ;put data in carry bit
                   0144          goto    NotOne
                   0145          xorlw   00h                     ;parity calculated by XORing all data bits
                   0146          goto    OneDone
                   0147 NotOne
                   0148          xorlw   01h
                   0149 OneDone
                   0150          decfsz  temp1
                   0151          goto    Parityloop              ;decrement count
                   0152          movwf   temp1
                   0153 ; Parity bit is in Bit 0 of temp1
                   0154 ;
                   0155     if _ODD_PARITY
                   0156          bsf     _parityBit
                   0157          btfsc   temp1,0
                   0158          bcf     _parityBit
                   0159     else
                   0160          bcf     _parityBit
                   0161          btfsc   temp1,0
```


© 1994 Microchip Technology Inc.

```
                0162          bsf     _parityBit
                0163      endif
                0164
                0165          return
                0166    endif
                0167 ;*****************************************************************************************************
                0287
                0288    include "rcvr.asm"     ; The Receiver Routines are in File "rcvr.asm"
                0001 ;*****************************************************************************
                0002 ;                       GetChar Function
                0003 ;  Receives a Byte Of Data
                0004 ;          When reception is complete, _rcvOver Bit is cleared
                0005 ;          The received data is in RxReg
                0006 ;
                0007 ;  Program Memory :        15 locations (17 locations if PARITY is used)
                0008 ;  Cycles     :     16 (18 if PARITY is USED)
                0009 ;
                0010 ;*****************************************************************************
                0011 GetChar:
0074 1283       0012          bcf     _rp0
0075 158F       0013          bsf     _rcvOver                ; Enable Reception, this bit gets reset on Byte Rcv Complete
                0014      LOAD_BITCOUNT
0076 3009        M            movlw   _DataBits+1
0077 0090        M            movwf   BitCount
0078 3001        M            movlw   1
0079 0091        M            movwf   ExtraBitCount
                 M       if  _PARITY_ENABLE
                 M            movlw   2
                 M            movwf   ExtraBitCount
                 M       endif
007A 018D       0015          clrf    RxReg
007B 128F       0016          bcf     _FrameErr
007C 120F       0017          bcf     _ParityErr              ; Init Parity & Framing Errors
007D 0181       0018          clrf    _rtcc
007E 0064       0019          clrwdt
007F 1683       0020          bsf     _rp0
0080 3007       0021          movlw   07h
0081 0081       0022          movwf   _option
0082 1283       0023          bcf     _rp0
0083 0181       0024          clrf    _rtcc
0084 1683       0025          bsf     _rp0
0085 300F       0026          movlw   0Fh
0086 0081       0027          movwf   _option
0087 0064       0028          clrwdt
0088 3038       0029          movlw   _OPTION_SBIT          ; Inc On Ext Clk Falling Edge
0089 0081       0030          movwf   _option               ; Set Option Reg Located In Page 1
008A 1283       0031          bcf     _rp0                  ; make sure to select Page 0
008B 30FF       0032          movlw   0xFF
```

© 1994 Microchip Technology Inc.

```
008C 0081          0033          movwf   _rtcc                ; A Start Bit will roll over RTCC & Gen INT
008D 110B          0034          bcf     _rtif
008E 168B          0035          bsf     _rtie                ; Enable RTCC Interrupt
008F 0009          0036          retfie                       ; Enable Global Interrupt
                   0037 ;
                   0038 ;*****************************************************************************************
                   0039 ;                  Internal Subroutine
                   0040 ; entered from Interrupt Service Routine when Start Bit Is detected.
                   0041 ;
                   0042 ;  Program Memory :       14 locations
                   0043 ;  Cycles    :    12 (worst case)
                   0044 ;
                   0045 ;*****************************************************************************************
                   0046 _SBitDetected:
0090 1283          0047          bcf     _rp0
0091 1A05          0048          btfsc   RX_Pin                       ; Make sure Start Bit Interrupt is not a Glitch
0092 28A5          0049          goto    _FalseStartBit               ; False Start Bit
0093 150F          0050          bsf     _rcvProgress
0094 0181          0051          clrf    _rtcc
0095 0064          0052          clrwdt
0096 1683          0053          bsf     _rp0
0097 3007          0054          movlw   07h
0098 0081          0055          movwf   _option
0099 1283          0056          bcf     _rp0
009A 0181          0057          clrf    _rtcc
009B 1683          0058          bsf     _rp0
009C 300F          0059          movlw   0Fh
009D 0081          0060          movwf   _option
009E 0064          0061          clrwdt
009F 3002          0062          movlw   (_BIT1_INIT | SBitPrescale)  ; Switch Back to INT Clock
00A0 0081          0063          movwf   _option                      ; Set Option Reg Located In Page 1
00A1 1283          0064          bcf     _rp0                         ; make sure to select Page 0
                   0065      LOAD_RTCC  1,(SBitRtccLoad), SBitPrescale
                     M       if(UsePrescale == 0 && 1 == 0)
                     M           movlw   -(SBitRtccLoad) + _Cycle_Offset1
                     M       else
00A2 3081            M           movlw   -(SBitRtccLoad) + (_Cycle_Offset1 >> (SBitPrescale+1))  ; Re Load RTCC init value + INT La
                     M       endif
00A3 0081            M           movwf   _rtcc           ; Note that Prescaler is cleared when RTCC is written
00A4 282D          0066          goto    RestoreIntStatus
                   0067 ;
                   0068 _FalseStartBit:
00A5 30FF          0069          movlw   0xFF
00A6 0081          0070          movwf   _rtcc                        ; reload RTCC with 0xFF for start bit detection
00A7 282D          0071          goto    RestoreIntStatus
                   0072 ;
                   0073 ;*****************************************************************************************
                   0074 ;                  Internal Subroutine
```


© 1994 Microchip Technology Inc.

```
                        0075 ; entered from Interrupt Service Routine when Start Bit Is detected.
                        0076 ;
                        0077 ;  Program Memory :       28 locations ( 43 locations with PARITY enabled)
                        0078 ;  Cycles     :     24 Worst Case
                        0079 ;
                        0080 ;*****************************************************************************************
                        0081 _RcvNextBit:
00A8 0064               0082         clrwdt
00A9 1683               0083         bsf     _rp0
00AA 3007               0084         movlw   07h
00AB 0081               0085         movwf   _option
00AC 1283               0086         bcf     _rp0
00AD 0181               0087         clrf    _rtcc
00AE 0064               0088         clrwdt
00AF 1683               0089         bsf     _rp0
00B0 3007               0090         movlw   07h
00B1 0081               0091         movwf   _option
00B2 1283               0092         bcf     _rp0
00B3 0181               0093         clrf    _rtcc
00B4 1683               0094         bsf     _rp0
00B5 300F               0095         movlw   0Fh
00B6 0081               0096         movwf   _option
00B7 0064               0097         clrwdt
00B8 3001               0098         movlw   (_OPTION_INIT | RtccPrescale)    ; Switch Back to INT Clock
00B9 0081               0099         movwf   _option                          ; Set Option Reg Located In Page 1
                        0100 ;
00BA 1283               0101         bcf     _rp0
00BB 0805               0102         movf    _porta,w                        ; read RX pin immediately into WREG
00BC 008E               0103         movwf   RxTemp
                        0104     LOAD_RTCC  0,RtccPreLoad, RtccPrescale      ; Macro to reload RTCC
                          M     if(UsePrescale == 0 && 0 == 0)
                          M         movlw   -RtccPreLoad + _Cycle_Offset1
                          M     else
00BD 3036                 M         movlw   -RtccPreLoad + (_Cycle_Offset1 >> (RtccPrescale+1))  ; Re Load RTCC init value + INT La
                          M     endif
00BE 0081                 M         movwf   _rtcc           ; Note that Prescaler is cleared when RTCC is written
00BF 0805               0105         movf    _porta,w
00C0 060E               0106         xorwf   RxTemp,w
00C1 3910               0107         andlw   RX_MASK         ; mask for only RX PIN (RA4)
00C2 1903               0108         btfsc   _z
00C3 28C6               0109         goto    _PinSampled     ; both samples are same state
                        0110 _SampleAgain:
00C4 0805               0111         movf    _porta,w
00C5 008E               0112         movwf   RxTemp          ; 2 out of 3 majority sampling done
                        0113 _PinSampled:
00C6 0890               0114         movf    BitCount,1
00C7 1903               0115         btfsc   _z
```

© 1994 Microchip Technology Inc.

```
00C8 28CB          0116          goto    _RcvP_Or_S
                   0117 ;
00C9 0B90          0118          decfsz  BitCount
00CA 28D1          0119          goto    _NextRcvBit
                   0120 ;
                   0121 _RcvP_Or_S:
                   0122    if _PARITY_ENABLE
                   0123          decfsz  ExtraBitCount
                   0124          goto    _RcvParity
                   0125    endif
                   0126 ;
                   0127 _RcvStopBit:
00CB 1E0E          0128          btfss   RX
00CC 168F          0129          bsf     _FrameErr             ; may be framing Error or Glitch
00CD 128B          0130          bcf     _rtie                 ; disable further interrupts
00CE 110F          0131          bcf     _rcvProgress
00CF 118F          0132          bcf     _rcvOver              ; Byte Received, Can RCV/TXMT an other Byte
                   0133    if _PARITY_ENABLE
                   0134          movf    RxReg,w
                   0135          call    GenParity             ; Generate Parity, for Parity check
                   0136          movlw   0
                   0137          btfsc   _parityBit
                   0138          movlw   0x10                  ; to mask off Received Parity Bit in _ParityErr
                   0139          xorwf   SerialStatus          ; _ParityErr bit is set accordingly
                   0140    endif
                   0141    if _DataBits == 7
                   0142          rrf     RxReg,1
                   0143          bcf     RxReg,7
                   0144    endif
00D0 282D          0145          goto    RestoreIntStatus
                   0146 ;
                   0147 _NextRcvBit:
00D1 1003          0148          bcf     _carry
00D2 1A0E          0149          btfsc   RX                    ; prepare bit for shift
00D3 1403          0150          bsf     _carry
00D4 0C8D          0151          rrf     RxReg                 ; shift in received data
00D5 282D          0152          goto    RestoreIntStatus
                   0153 ;
                   0154    if _PARITY_ENABLE
                   0155 _RcvParity:
                   0156          bcf     _ParityErr            ; Temporarily store PARITY Bit in _ParityErr
                   0157          btfsc   RX                    ; Sample again to avoid any glitches
                   0158          bsf     _ParityErr
                   0159          goto    RestoreIntStatus
                   0160    endif
                   0161 ;
                   0162 ;*************************************************************************************
                   0288
```


© 1994 Microchip Technology Inc.

```
                    0289
                    0290 ;*****************************************************************************************************************
                    0291
                    0292         END
                    0293
                    0294
                    0295
                    0296


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXX------ -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN554

Software Implementation of I²C™ Bus Master

INTRODUCTION

This application note describes the software implementation of I²C™ interface routines for the PIC16CXX family of devices. Only the master mode of I²C interface is implemented. This implementation is for a single master communication to multiple slave I²C devices.

Some PIC16CXX devices, such as the PIC16C64 and PIC16C74, have on-chip hardware which implements the I²C slave interface, while other PIC16CXX devices, such as the PIC16C71 and PIC16C84, do not have any on-chip hardware.

This application note does not describe the I²C Bus specifications and the user is assumed to have an understanding of the I²C Bus. For detailed information on the bus, the user is advised to read the I²C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I²C Bus is a two-wire serial bus with multiple masters and multiple slaves connected to each other through two wires. The two wires consists of a clock line (SCL) and a data line (SDA) with both the lines being bi-directional. Bi-directional communication is facilitated through the use of wire-and connection (the lines are either active-low or passive high). The I²C Bus protocol also allows collision detection, clock synchronization and hand-shaking for multi-master systems. The clock is always generated by the master, but the slave may hold it low to generate a wait state.

In most systems the microcontroller is the master and the external peripheral devices are slaves. In these cases this application note can be used to attach I²C slaves to the PIC16CXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I²C Bus timing specifications using software. For a true slave or multi-master system, some interface hardware is necessary (like START & STOP bit detection).

In addition to the low-level single master I²C routines, a collection of high level routines with various message structures is given. These high level macros/routines can be used as canned routines to interface to most I²C Slave devices. As an example, the test program talks to two Serial EEPROMs (Microchip's 24LC04 & 24LC01).

IMPLEMENTATION

Two levels of software routines are provided. The low-level routines are given in "i2c_low.asm" and the high level routines are given in "i2c_high.asm". The routines are described later. The messages passed (communicated on the two wire network) are abbreviated and certain notation is used to represent Start, Stop and other conditions. These abbreviations are described at first in Table 1.

TABLE 1 - DESCRIPTION OF ABBREVIATIONS USED

Abbreviation	Explanation
S	Start Condition
P	Stop Condition
SlvAR	Slave Address (for read operation)
SlvAW	Slave Address (for write operations)
A	Acknowledge condition (positive ACK)
N	Negative Acknowledge condition (NACK)
D	Data byte, D[0] represents byte 0, D[1] represents second byte

© 1994 Microchip Technology Inc.

Message Format

In the high level routines, the basic structure of the message implemented is given. Every I²C slave supports one or more message structures. For example, Microchip's 24LC04 Serial EEPROM supports the following message (to write a byte to Serial EEPROM at current address counter) S-SlvAW-A-D-A-P which basically means the following sequence of operations are required :

(a) Send Start Bit

(b) Send Slave Address for Write Operations

(c) Expect Acknowledge

(d) Send Data Byte

(e) Expect Acknowledge

(f) Issue a STOP Condition

Slave Address

Both 10-bit and 7-Bit addressing schemes are implemented as specified by the I²C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD_ADDR_8" (for 7-bit address slaves) or "LOAD_ADDR_8" macro (for 10-bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7- or 10-bit addressing modes. See the macros section for more details.

CLOCK STRETCHING

In I²C Bus, the clock (SCL Line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built in wait state for the I²C Bus. This feature is implemented and can be turned on or off as an assembly time option (by setting _ENABLE_BUS_FREE_TIME flag to be TRUE or FALSE). If the clock is held low for too long, say 1 msec, then an error condition is assumed and an RTCC interrupt is generated.

ARBITRATION

The I²C Bus specifies both bit-by-bit and byte mode arbitration procedures for multi-master systems. However, the arbitration is not needed in a single master system, and therefore not implemented in this application note.

HARDWARE

Two I/O pins are used to emulate the Clock Line SCL and the data line SDA. In the example test program, RB0 is used as SCL and RB1 as SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with 0s. To emulate the high state (passive), these lines are turned as inputs and to emulate the active low state, the pins are turned as outputs (with the assumption of having external pull-up resistors on both the lines).

For devices that have the on-chip I²C hardware (SSP module), slope control of the I/O is implemented on the SCK and SDA pins. For software not implemented on the SCK and SDA pins of the SSP module, external components for slope control of the I/O may be required by the system.

© 1994 Microchip Technology Inc.

I²C ROUTINES

Status Register (File Register "Bus_Status") :

The bit definitions of the status register are described in the table given below. These bits reflect the status of the I²C Bus.

Bit #	Name	Description
0	_Bus_Busy	1 = Start Bit transmitted 0 = STOP condition.
1	_Abort	It is set when a fatal error condition is detected. The user must clear this bit. This bit is set when Clock Line (SCL) is stuck low.
2	_Txmt_Progress	1 = transmission in progress.
3	_Rcv_Progress	1 = reception in progress.
4	_Txmt_Success	1 = transmission succesfully completed. 0 = error condition.
5	_Rcv_Success	1 = reception succesfully completed. 0 = error condition.
6	_Fatal_Error	1 = FATAL error occurred. The communication was aborted.
7	$\overline{\text{ACK}}$ _Error	1 = slave sent not $\overline{\text{ACK}}$ while the master was expecting an $\overline{\text{ACK}}$. This may happen for example if the slave was not responding to a message.

Control Register (File Register "Bus_Control") :

The bit definitions of the control register are described in the table given below. These bits must be set by the software prior to performing certain operations. Some of the high level routines described later in this section set these bits automatically.

Bit #	Name	Description
0	_10BitAddr	1 = 10 bit slave addressing 0 = 7 bit addressing.
1	_Slave_RW	1 = READ operation 0 = WRITE operation.
2	_Last_Byte_Rcv	1 = last byte must be received. Used to send not $\overline{\text{ACK}}$.
3,4,5	—	Unused bits, can be used as general purpose bits.
6	_SlaveActive	A status bit indicating if a slave is responding. This bit is set or cleared by calling the I²C_TEST_DEVICE macro. See description of this I²C_TEST_DEVICE macro.
7	_TIME_OUT_	A status bit indicating if a clock is stretched low for more than 1 ms, indicationg a bus error. On this time out, the operation is aborted.

© 1994 Microchip Technology Inc.

Low Level :

Function Name	Description
InitI²CBus_Master	Initializes Control/Status Registers, and set SDA & SCL lines. Must be called on initialization.
TxmtStartBit	Transmits a START (S) condition.
TxmtStopBit	Transmits a STOP (P) condition.
LOAD_ADDR_8	The 7 bit slave's address must be passed as a constant parameter.
LOAD_ADDR_10	The 10 bit slave's address must be passed as a constant parameter.
Txmt_Slave_Addr	Transmits a Slave address. Prior to calling this routine, the address of the slave being addressed must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 routines. Also the Read/Write condition must be set in the control register.
SendData	Transmits a byte of data. Prior to calling this routine, the byte to be transmitted must be loaded into DataByte file register.
GetData	Receives a byte of data in DataByte file register. If the data byte to be received is the last byte, the _Last_Byte_Rcv bit in control register must be set priot to calling this routine.

© 1994 Microchip Technology Inc.

MACROS

High Level :

The high level routines are implemented as a mixture of function calls and macros. These high level routines call the low level routines described above. In most cases only a few of the high level routines may be used and the user can remove or not include the routines not necessary to conserve program memory space. Examples are given for a few functions.

I²C_TEST_DEVICE

Parameters : None

Purpose : To test if a slave is present on the network

Description : Before using this macro, the address of the slave being tested must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 macro. Is the slave under test is present, then "_SlaveActive" status bit (in Bus_Control file register) is set. If not, then this bit is set 0, indicating that the slave is either not present on the network or is not listening.

Message : *S-SlvAW-A-P*

Example :

```
LOAD_ADDR_8    0xA0              ; 24LC04 address
I²C_TEST_DEVICE
btfss          _SlaveActive      ; See If slave is responding
goto           SlaveNotPresent   ; 24LC04 is not present
                                 ; Slave is present
                                 ; Continue with program
```

I²C_WR

Parameters : _BYTES_, _SourcePointer_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file registers)

Purpose : A basic macro for writing a block of data to a slave

Description : This macro writes a block of data (no of bytes = _BYTES_) to a slave. The starting address of the block of data is _SourcePointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Txmt_Success bit)

Message : *S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P*

Example :

```
btfsc      _Bus_Busy       ; Check if bus is free
goto       $-1
LOAD_ADDR_8      _Slave_1_Addr
I²C_WR           0x09, DataBegin ;
```

I²C_WR_SUB

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as I²C_WR function, except that the starting address of the slave is also specified. For example, while writing to an I²C Memory Device, the sub-addr specifies the starting address of the memory. The I²C_WR may prove to be more efficient than this macro in most situations. Advantages will be found for Random Address Block Writes for Slaves with Auto Increment Sub-addresses (like Microchip's 24CXX series Serial EEPROMs)

Message : *S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P*

Example :

```
LOAD_ADDR_8  _Slave_2_Addr          ; Load addr of 7 bit slave
I²C_WR_SUB   0x08, DataBegin+1, 0x30
```

In the above example , 8 Bytes of data starting from addr (DataBegin+1) is written to 24LC04 Serial EEPROM beginning at 0x30 address

I²C_WR_SUB_SWINC

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as I²C_WR_SUB function, except that the sub-address (incremented) is sent after every data byte. A very inefficient message structure and the Bus is given up after each data byte. This is useful for when the slave does not have an auto-increment sub-address feature.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
and so on until #of Bytes

© 1994 Microchip Technology Inc.

I²C_WR_BYTE_MEM

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-address

Description : Same as I²C_WR_SUB_SWINC, except that a delay is added between each message. This is necessary for some devices like EEPROMs which accept only a byte at a time for programming (devices without on chip ram buffer) and after each byte a delay is necessary before a next byte is written.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*
Delay 1 msec
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
Delay 1 msec
and so on until #of Bytes

I²C_WR_BUF_MEM

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_, _Device_BUF_SIZE_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave
_Device_BUF_SIZE_	the slaves on-chip buffer size

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : This Macro/Function writes #of _BYTES_ to an I²C memory device. However some devices, especially EEPROMs, must wait while the device enters into programming mode. But some devices have an on-chip temperature data hold buffer and is used to store data before the device actually enters into programming mode. For example, the 24C04 series of Serial EEPROMs from Microchip have an 8-byte data buffer. So one can send 8 bytes of data at a time and then the device enters programming mode. The master can either wait until a fixed time and then retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming.

Message : I²C_SUB_WR operations are performed in loop and each time data buffer of BUF_SIZE is output to the device. Then the device is checked for busy and when not busy another block of data is written.

© 1994 Microchip Technology Inc.

I²C_READ

Parameters : _BYTES_, _DestPointer_

BYTES Number of bytes starting from RAM pointer _SourcePointer_

DestPointer Data Start Buffer pointer in RAM (file Registers)

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (number of bytes = _BYTES_) from a slave. The starting address of the block of data is _DestPointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Rcv_Success bit). Note that on the last byte to receive, NACK is sent.

Message : *S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10  _Slave_3_Addr
I²C_READ 8, DataBegin
 btfss     _Rcv_Success
 goto      ReceiveError
 goto      ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10-bit slave and stored in the master's ram starting at address DataBegin.

© 1994 Microchip Technology Inc.

I²C_READ_SUB

Parameters : _BYTES_, _DestPointer_, _SubAddress

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
DestPointer	Data Start Buffer pointer in RAM (file Registers)
SubAddress	Sub-address of the slave

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (no of bytes = _BYTES_) from a slave starting at slave's sub-address _SubAddress. The data received is stored in master's ram starting at address _DestAddress. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Rcv_Success bit).

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address between the sub-address write and the following reads, a STOP condition is not issued and a "REPEATED START" condition is used so that an other master will not take over the bus, and also that no other master will overwrite the sub-address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a memory device (e.g. : 24CXX serial of Serial EEPROMs from Microchip).

Message : *S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10  _Slave_3_Addr
I²C_READ_SUB  8, DataBegin, 0x60
btfss     _Rcv_Success
goto      ReceiveError
goto      ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave (starting at address 0x60) and stored in the master's ram starting at address DataBegin.

I²C_READ_BYTE or I²C_READ_STATUS

Parameters : _DestPointer_

DestPointer	Data Start Buffer pointer in RAM (file Registers)

Purpose : To read a Status Byte from Slave

Description : Several I²C Devices can send a Status Byte upon reception of the control byte. This Macro reads a Status byte from a slave to the master's RAM location _DestPointer_. This function is basically the same as I²C_READ for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when I²C_READ_STATUS function is called. On successful operation of this command, WREG = 1 else WREG = 0 on errors.

Message : *S-SlvAR-A-D-A-N-P*

© 1994 Microchip Technology Inc.

I²C_WR_SUB_WR

Parameters : _Bytes1_, _SrcPtr1_, _SubAddr_, _Bytes2_, _SrcPtr2_

Bytes1	Number of bytes in the first data block
SrcPtr1	Starting address of first data block
SubAddr	Sub-address of the slave
Bytes2	Number of bytes in the second data block
SrcPtr2	Starting address of second data block

Purpose : To send two blocks of data to a slave at its sub address

Description : This Macro writes two blocks of data (of variable length) starting at slave's sub-address _SubAddr_. This Macro is essentially the same as calling I²C_WR_SUB twice, but a STOP bit is not sent in between the transmission of the two blocks. This way the Bus is not given up.

This function may be useful for devices which need two blocks of data in which the first block may be an extended address of a slave device. For example, a large I²C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the first block that represents the actual physical address and is followed by a second block that actually represents the data.

Message : *S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P*

I²C_WR_SUB_RD

Parameters : _Count1_, _SrcPtr_, _SubAddr_, _Count2_, _DestPtr_

Count1	Length of the source buffer
SrcPtr	Source pointer address
SubAddr	Sub-address of the slave
Count2	Length of the destination buffer
DestPtr	Address of destination buffer

Purpose : To send a block of data and then receive a block of data

Description : This macro writes a block of data (of length _Count1_) to a slave starting at sub-address _SubAddr_ and then reads a block of data (of length _Count2_) to the master's destination buffer (starting at address _DestPtr_). Although this operation can be performed using previously defined Macros, this function does not give up the bus in between the block writes and block reads. This is achieved by using the Repeated Start Condition.

Message : *S-SlvW-A-SubA-A-D1[0]-A-.....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P*

© 1994 Microchip Technology Inc.

I²C_WR_COM_WR

Parameters : _Count1_, _SrcPtr1_, _Count2_, _SrcPtr2_

Count1	Length of the first data block
SrcPtr1	Source pointer of the first data block
Count2	Length of the second data block
SrcPtr2	Source pointer of the second data block

Purpose : To send two blocks of data to a slave in one message.

Description : This macro writes a block of data (of length _Count1_) to a slave and then sends another block of data (of length _Count2_) without giving up the bus.

For example, this kind of transaction can be used in an I²C LCD driver where a block of control and address information is needed and then another block of actual data to be displayed is needed.

Message : *S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P*

3

APPLICATIONS

The I²C Bus is a simple two wire serial protocol for inter-IC communications. Many peripheral devices (acting as slaves) are available in the market with I²C interface (e.g. serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters, etc.). Although some of the PIC16CXX devices do not have on chip I²C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I²C bus can be implemented using software.

Author: Amar Palacherla,
Logic Products Division

© 1994 Microchip Technology Inc.

Appendix A - I²C.H

```
;*************************************************************************************************************
;                       I2C Bus Header File
;*************************************************************************************************************

_ClkOut         equ     (_ClkIn >> 2)

;
; Compute the delay constants for setup & hold times
;
_40uS_Delay     set     (_ClkOut/250000)
_47uS_Delay     set     (_ClkOut/212766)
_50uS_Delay     set     (_ClkOut/200000)

#define _OPTION_INIT    (0xC0 | 0x03)          ; Prescaler to RTCC for Appox 1 mSec timeout
;
#define _SCL    _       portb,0
#define _SDA    _       portb,1

#define _SCL_TRIS       trisb,0
#define _SDA_TRIS       trisb,1

#define _WRITE_         0
#define _READ_          1


;                               Register File Variables


        CBLOCK  0x0C
                SlaveAddr               ; Slave Addr must be loader into this reg
                SlaveAddrHi             ; for 10 bit addressing mode
                DataByte                ; load this reg with the data to be transmitted
                BitCount                ; The bit number (0:7) transmitted or received
                Bus_Status              ; Status Reg of I2C Bus for both TXMT & RCVE
                Bus_Control             ; control Register of I2C Bus
                DelayCount
                DataByteCopy            ; copy of DataByte for Left Shifts (destructive)
                SubAddr                 ; sub-address of slave (used in I2C_HIGH.ASM)
                SrcPtr                  ; source pointer for data to be transmitted
                tempCount               ; a temp variable for scratch RAM
                StoreTemp_1             ; a temp variable for scratch RAM, do not disturb contents
                End_I2C_Ram             ; unused, only for ref of end of RAM allocation
                 ENDC
```


© 1994 Microchip Technology Inc.

```
;*******************************************************************************************************
;               I2C Bus Status Reg Bit Definitions
;*******************************************************************************************************

#define _Bus_Busy      Bus_Status,0
#define _Abort         Bus_Status,1
#define _Txmt_Progress Bus_Status,2
#define _Rcv_Progress  Bus_Status,3

#define _Txmt_Success  Bus_Status,4
#define _Rcv_Success   Bus_Status,5
#define _Fatal_Error   Bus_Status,6
#define _ACK_Error     Bus_Status,7

;*******************************************************************************************************
;               I2C Bus Contro Register
;*******************************************************************************************************
#define _10BitAddr     Bus_Control,0
#define _Slave_RW      Bus_Control,1
#define _Last_Byte_Rcv Bus_Control,2

#define _SlaveActive   Bus_Control,6
#define _TIME_OUT_     Bus_Control,7



;*******************************************************************************************************
;                       General Purpose Macros
;*******************************************************************************************************

RELEASE_BUS    MACRO
               bsf     _rp0           ; select page 1
               bsf     _SDA           ; tristate SDA
               bsf     _SCL           ; tristate SCL
;              bcf     _Bus_Busy      ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
       ENDM

;*******************************************************************************************************
;               A MACRO To Load 8 OR 10 Bit Address To The Address Registers
;
;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
;  on 8 or 10 bit addressing modes
;*******************************************************************************************************

LOAD_ADDR_10   MACRO   SLAVE_ADDRESS

       bsf     _10BitAddr                              ; Slave has 10 bit address
```


© 1994 Microchip Technology Inc.

```
        movlw   (SLAVE_ADDRESS & 0xff)
        movwf   SlaveAddr                               ; load low byte of address
        movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)  ; 10 bit addr is 11110XX0
        movwf   SlaveAddr+1                             ; hi order address

        ENDM
LOAD_ADDR_8     MACRO   SLAVE_ADDRESS

        bcf     10BitAddr                               ; Set for 8 Bit Address Mode
        movlw   (SLAVE_ADDRESS & 0xff)
        movwf   SlaveAddr

                ENDM
```


© 1994 Microchip Technology Inc.

Appendix B - TEST.ASM

```
  Title         "I2C Master Mode Implemetation"
        SubTitle        "Rev 0.1      :  01 Mar 1993"

;*****************************************************************************************************
;
;               Software Implementation Of I2C Master Mode
;
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
;
; *     Refer to Signetics/Philips I2C-Bus Specification
;
;    The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
;
; RB1 is SDA            (Any I/O Pin May Be used instead)
; RB0/INT is SCL        (Any I/O Pin May Be used instead)
;
;
;*****************************************************************************************************

        Processor       16C71
        Radix   DEC

_ClkIn          equ     16000000        ; Input Clock Frequency Of PIC16C71

        include         "d:\pictools\16Cxx.h"
;
#define _Slave_1_Addr   0xA0            ; Serial EEPROM #1
#define _Slave_2_Addr   0xAC            ; Serial EEPROM #2
#define _Slave_3_Addr   0xD6            ; Slave PIC16CXX

#define _ENABLE_BUS_FREE_TIME  TRUE
#define _CLOCK_STRETCH_CHECK   TRUE
#define _INCLUDE_HIGH_LEVEL_I2C         TRUE


        include         "i2c.h"

        CBLOCK  End_I2C_Ram
                SaveStatus              ; copy of STATUS Reg
                SaveWReg                ; copy of WREG
                byteCount
                HoldData
        ENDC

        CBLOCK  0x20
```

© 1994 Microchip Technology Inc.

```
                DataBegin               ; Data to be read or written is stored here
                 ENDC

        ORG     0x00

        goto    Start
;
        ORG     0x04
;*****************************************************************************************************
;                       Interrupt Service Routine
;
;   For I2C routines, only RTCC interrupt is used
; RTCC Interrupts enabled only if Clock Stretching is Used
; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
; The user may choose to retry the operation later again
;
;*****************************************************************************************************

Interrupt:
;
; Save Interrupt Status (WREG & STATUS regs)
;
  movwf         SaveWReg                ; Save WREG
  swapf _       status,w                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
  movwf         SaveStatus              ; Save STATUS Reg
 if _CLOCK_STRETCH_CHECK                ; RTCC Interrupts enabled only if Clock Stretching is Used
  btfss _       rtif
  goto          MayBeOtherInt           ; other Interrupts
  bsf   _       TIME_OUT_               ; MUST set this Flag, can take other desired actions here
  bcf   _       rtif
 endif
;
;  Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
;
MayBeOtherInt:
  NOP
;
RestoreIntStatus:                       ; Restore Interrupt Status
  swapf         SaveStatus,w
  movwf _       status                  ; restore STATUS Reg
  swapf         SaveWReg
  swapf         SaveWReg,w              ; restore WREG
  retfie
;
;*****************************************************************************************************
;                       Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
      include       "i2c_high.asm"
```


© 1994 Microchip Technology Inc.

```
  endif
;*****************************************************************************************************
;
ReadSlave1:
;
;  EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
;
       LOAD_ADDR_8  _Slave_1_Addr
wait1:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive     ; See If slave is responding
       goto            wait1           ; if stuck for ever, recover from WDT, can use other schemes
       clrwdt
       I2C_READ_SUB  8, DataBegin+1, 0x50
;
; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
;
       LOAD_ADDR_8  _Slave_2_Addr
wait2:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive     ; See If slave is responding
       goto            wait2           ; if stuck for ever, recover from WDT, can use other schemes
       clrwdt
       I2C_READ_SUB  8, DataBegin+1, 0x60


  return
;
;*****************************************************************************************************

ReadSlave3:

       LOAD_ADDR_8  _Slave_3_Addr
wait3:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive     ; See If slave is responding
       goto            wait3           ; if stuck for ever, recover from WDT, can use other schemes
       clrwdt
       I2C_READ_SUB  8, DataBegin, 0
;

  return

;*****************************************************************************************************
;
;             Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
```


© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************************************

FillDataBuf:

  movlw         0x00                    ; start address location of EEPROM array
  movwf         DataBegin               ; 1st byte of data to be sent is start address
  movlw         DataBegin+1             ; data starts following address (RAM Pointer)
  movwf _       fsr
  movlw         8                       ; fill RAM with 8 bytes , this data is written to EEPROM (slave)
  movwf         byteCount
  movlw         0x55                    ; pattern to fill with is 0x55 & 0xAA
  movwf         HoldData
X1:
  comf          HoldData
  movf          HoldData,w
  movwf _       indf
  incf  _       fsr                     ; point to next location
  decfsz        byteCount
  goto          X1
  return
;
;*****************************************************************************************************
;
;           Main Routine  (Test Program)
;
;           SINGLE  MASTER, MULTIPLE SLAVES
;
;*****************************************************************************************************

Start:
  call          InitI2CBus_Master       ; initialize I2C Bus
  bsf   _       gie                     ; enable global interrupts
;

  call  FillDataBuf                     ; fill data buffer with 8 bytes of data (0x55, 0xAA)
;
; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit Addr
;
;  Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
;

  btfsc _       Bus_Busy                ; is Bus Free, ie. has a start & stop  bit been detected (only for multi master system)
  goto          $-1                     ; a very simple test, unused for now

  LOAD_ADDR_8  _Slave_1_Addr
  I2C_WR              0x09, DataBegin
;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
```


© 1994 Microchip Technology Inc.

```
;
  btfsc         Bus_Busy        ; is Bus Free, ie. has a start & stop bit been detected (only for multi master system)
  goto          $-1             ; a very simple test, unused for now

  LOAD_ADDR_8   _Slave_2_Addr
  I2C_WR_SUB     0x08, DataBegin+1, 0x30

  call  ReadSlave1              ; read a byte from slave from current address
;
  LOAD_ADDR_8   _Slave_3_Addr
  movlw 0xCC
        movwf   DataBegin
  I2C_WR_SUB    0x01,DataBegin, 0x33
;
  call  ReadSlave3              ; Read From Slave PIC
;

self    clrwdt
  goto  self
;
;*****************************************************************************************************************

  END
```


© 1994 Microchip Technology Inc.

Appendix C - LOW.ASM

```
;*******************************************************************************************************
;
;                       Low Level I2C Routines
;
; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
;    when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
;    detection is available.
;
;   The generic high level routines are given in I2C_HIGH.ASM
;
;*******************************************************************************************************



;*******************************************************************************************************
;                       I2C Bus Initialization
;
;*******************************************************************************************************
InitI2CBus_Master:

  bcf   _       rp0
  movf  _       portb,w
  andlw         0xFC            ; do not use BSF & BCF on Port Pins
  movwf _       portb           ; set SDA & SCL to zero. From Now on, simply play with tris
  RELEASE_BUS
  clrf          Bus_Status      ; reset status reg
  clrf          Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
  return
;
;*******************************************************************************************************
;                               Send Start Bit
;
;*******************************************************************************************************

TxmtStartBit:
       bsf     _rp0            ; select page 1
       bsf     _SDA            ; set SDA high
       bsf     _SCL            ; clock is high
;
; Setup time for a REPEATED START condition (4.7 uS)
;
       call    Delay40uSec     ; only necesry for setup time
;
```


© 1994 Microchip Technology Inc.

```
        bcf     _SDA            ; give a falling edge on SDA while clock is high
;
        call    Delay47uSec     ; only necessary for START HOLD time
;
        bsf     _Bus_Busy       ; on a start condition bus is busy
;
        return


;*****************************************************************************************************
;                               Send Stop Bit
;
;*****************************************************************************************************

TxmtStopBit:
        bsf     _rp0            ; select page 1
        bcf     _SCL
        bcf     _SDA            ; set SDA low
        bsf     _SCL            ; Clock is pulled up
        call    Delay40uSec     ; Setup Time For STOP Condition
        bsf     _SDA            ; give a rising edge on SDA while CLOCK is high
;
  if _ENABLE_BUS_FREE_TIME
; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
;
        call    Delay47uSec
  endif
;
        bcf     _Bus_Busy       ; on a stop condition bus is considered Free
;
        return

;*****************************************************************************************************
;                               Abort Transmission
;
;   Send STOP Bit & set Abort Flag
;*****************************************************************************************************

AbortTransmission:

        call    TxmtStopBit
        bsf     _Abort
        return

;*****************************************************************************************************
;                       Transmit Address (1st Byte)& Put in Read/Write Operation
;
;  Transmits Slave Addr On the 1st byte and set LSB to R/W operation
```


© 1994 Microchip Technology Inc.

```
;  Slave Address must be loaded into SlaveAddr reg
;  The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
;
;  On Success, return TRUE in WREG, else FALSE in WREG
;
;   If desired, the failure may tested by the bits in Bus Status Reg
;
;*****************************************************************************************************

Txmt_Slave_Addr:
  bcf           ACK_Error               ; reset Acknowledge error bit
  btfss         10BitAddr
  goto          SevenBitAddr
;
  btfss         Slave_RW
  goto          TenBitAddrWR            ; For 10 Bit WR simply send 10 bit addr
;
; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
; and then Hi Byte Only for read opreation
;
TenBitAddrRd:

  bcf           Slave_RW                ; temporarily set for WR operation
  call          TenBitAddrWR
  btfss         Txmt_Success            ; skip if successful
  retlw FALSE

  call          TxmtStartBit            ; send A REPEATED START condition
  bsf           Slave_RW                ; For 10 bit slave Read

  movf          SlaveAddr+1,W
  movwf         DataByte
  bsf           DataByte,LSB            ; Read Operation
  call          SendData                ; send ONLY high byte of 10 bit addr slave
  goto _        AddrSendTest            ; 10 Bit Addr Send For Slave Read Over
;
; if successfully transmitted, expect an ACK bit
;
  btfss _       Txmt_Success            ; if not successful, generate STOP & abort transfer
  goto _        AddrSendFail
;

TenBitAddrWR:

  movf          SlaveAddr+1,W
  movwf         DataByte
  bcf           DataByte,LSB            ; WR Operation
;
; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
```


© 1994 Microchip Technology Inc.

```
; then save RETURN Address Pointer
;
  call          SendData                ; send high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
  btfss         Txmt_Success            ; if not successful, generate STOP & abort transfer
  goto          AddrSendFail
;
  movf          SlaveAddr,W
  movwf         DataByte                ; load addr to DatByte for transmission
  goto          EndTxmtAddr

SevenBitAddr:
  movf          SlaveAddr,W
  movwf         DataByte                ; load addr to DatByte for transmission
  bcf           DataByte,LSB
  btfsc _       Slave_RW                ; if skip then write operation
  bsf           DataByte,LSB            ; Read Operation

EndTxmtAddr:
  call          SendData                ; send 8 bits of address, bus is our's
;
; if successfully transmitted, expect an ACK bit
;
_AddrSendTest:
  btfss _       Txmt_Success            ; skip if successful
  goto          AddrSendFail
  clrwdt
  retlw         TRUE
;
_AddrSendFail:
  clrwdt
  btfss _       ACK_Error
  retlw         FALSE                   ; Addr Txmt Unsuccessful, so return 0
;
; Address Not Acknowledged, so send STOP bit
;
  call          TxmtStopBit
  retlw         FALSE                   ; Addr Txmt Unsuccessful, so return 0
;
;*************************************************************************************************************
;                       Transmit A Byte Of Data
;
; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
; and INT Interrupt on Rising edge is enabled and also RTCC timeout interrupt is enabled
; The clock stretching slows down the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock stretching, then
```

© 1994 Microchip Technology Inc.

```
; this feature can be disabled during Assembly time by setting
; _CLOCK_STRETCH_ENABLED must be set to FALSE.
;
;*************************************************************************************************
SendData:

;
; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
; For future compatibility, the user MUST call SendData & NOT TxmtByte
;
  goto  TxmtByte

;
TxmtByte:
        movf    DataByte,w
        movwf   DataByteCopy            ; make copy of DataByte
        bsf     Txmt_Progress           ; set Bus status for txmt progress
        bcf     Txmt_Success            ; reset status bit
        movlw   0x08
        movwf   BitCount
        bsf     _rp0
    if _CLOCK_STRETCH_CHECK
;  set RTCC to INT CLK timeout for 1 mSec
;  do not disturb user's selection of RPUB in OPTION Register
;
        movf    option,w
        andlw   OPTION_INIT             ; defined in I2C.H header file
        movwf   option
    endif

TxmtNextBit:
        clrwdt                          ; clear WDT, set for 18 mSec
        bcf     SCL
        rlf     DataByteCopy            ; MSB first, Note DataByte Is Lost
                bcf     _SDA
        btfsc   c
        bsf     SDA
        call    Delay47uSec             ; guareentee min LOW TIME tLOW & Setup time
        bsf     SCL                     ; set clock high , check if clock is high, else clock being stretched
        call    Delay40uSec             ; guareentee min HIGH TIME tHIGH
  if _CLOCK_STRETCH_CHECK
        bcf     rp0
        clrf    rtcc                    ; clear RTCC
        bcf     rtif                    ; clear any pending flags
        bsf     rtie                    ; elable RTCC Interrupt
        bcf     TIME_OUT_               ; reset timeout error flag
Check_SCL_1:
        btfsc   TIME_OUT_               ; if RTCC timeout or Error then Abort & return
        goto    Bus_Fatal_Error         ; Possible FATAL Error on Bus
```


© 1994 Microchip Technology Inc.

```
        bcf     rp0
        btfss   SCL             ; if clock not being stretched, it must be high
        goto    Check_SCL_1     ; loop until SCL high or RTCC timeout interrupt
        bcf     rtie            ; Clock good, diable RTCC interrupts
        bsf     rp0
  endif
        decfsz  BitCount
        goto    TxmtNextBit
;
; Check For Acknowledge
;
        bcf     SCL             ; reset clock
        bsf     SDA             ; Release SDA line for Slave to pull down
        call    Delay47uSec     ; guareentee min LOW TIME tLOW & Setup time
        bsf     SCL             ; clock for slave to ACK
        call    Delay40uSec     ; guareentee min HIGH TIME tHIGH
        bcf     rp0             ; select PAGE 0 to test PortB pin SDA
        btfsc   SDA             ; SDA should be pulled low by slave if OK
                goto    _TxmtErrorAck
;
        bsf     rp0
        bcf     SCL             ; reset clock

        bcf     Txmt_Progress   ; reset TXMT bit in Bus Status
        bsf     Txmt_Success    ; transmission successful
        bcf     ACK_Error       ; ACK OK
        return
_TxmtErrorAck:
        RELEASE_BUS
        bcf     Txmt_Progress   ; reset TXMT bit in Bus Status
        bcf     Txmt_Success    ; transmission NOT successful
        bsf     ACK_Error       ; No ACK From Slave
        return
;
;*****************************************************************************************************
;
;                       Receive  A Byte Of Data From Slave
;
;  assume address is already sent
;  if last byte to be received, do not acknowledge slave (last byte is testted from
;  _Last_Byte_Rcv bit of control reg)
;  Data Received on successful reception is in DataReg register
;
;
;*****************************************************************************************************
;
GetData:
        goto    RcvByte
```

© 1994 Microchip Technology Inc.

```
;
RcvByte:

        bsf     Rcv_Progress            ; set Bus status for txmt progress
        bcf     Rcv_Success             ; reset status bit

        movlw   0x08
        movwf   BitCount
  if _CLOCK_STRETCH_CHECK
        bsf     _rp0
;  set RTCC to INT CLK timeout for 1 mSec
;  do not disturb user's selection of RPUB in OPTION Register
;
        movf    option,w
        andlw   OPTION_INIT             ; defined in I2C.H header file
        movwf   option
  endif

RcvNextBit:
        clrwdt                          ; clear WDT, set for 18 mSec
        bsf     rp0                     ; page 1 for TRIS manipulation
        bcf     SCL
        bsf     SDA                     ; can be removed from loop
        call    Delay47uSec             ; guareentee min LOW TIME tLOW & Setup time
        bsf     SCL                     ; clock high, data sent by slave
        call    Delay40uSec             ; guareentee min HIGH TIME tHIGH
  if _CLOCK_STRETCH_CHECK
        bcf     rp0
        clrf    rtcc                    ; clear RTCC
        bcf     rtif                    ; clear any pending flags
        bsf     rtie                    ; elable RTCC Interrupt
        bcf     TIME_OUT_               ; reset timeout error flag
Check_SCL_2:
        btfsc   TIME_OUT_               ; if RTCC timeout or Error then Abort & return
        goto    Bus_Fatal_Error         ; Possible FATAL Error on Bus
        bcf     rp0
        btfss   SCL                     ; if clock not being stretched, it must be high
        goto    Check_SCL_2             ; loop until SCL high or RTCC timeout interrupt
        bcf     rtie                    ; Clock good, diable RTCC interrupts
        bsf     rp0
  endif
        bcf     rp0                     ; select page 0 to read Ports
        bcf     c
        btfsc   SDA
        bsf     c
;                                       ; TEMP ???? DO 2 out of 3 Majority detect
         rlf    DataByte                ; left shift data ( MSB first)
        decfsz  BitCount
        goto    RcvNextBit
```


© 1994 Microchip Technology Inc.

```
;
; Generate ACK bit if not last byte to be read,
; if last byte Gennerate NACK        ; do not send ACK on last byte, main routine will send a STOP bit
;
       bsf     rp0
       bcf     SCL
       bcf     SDA                   ; ACK by pulling SDA low
       btfsc   Last_Byte_Rcv
       bsf     SDA                   ; if last byte, send NACK by setting SDA high
       call    Delay47uSec           ; guarantee min LOW TIME tLOW & Setup time
       bsf     SCL
       call    Delay40uSec           ; guarantee min HIGH TIME tHIGH
RcvEnd:
       bcf     SCL                   ; reset clock

       bcf     Rcv_Progress          ; reset TXMT bit in Bus Status
       bsf     Rcv_Success           ; transmission successful
       bcf     ACK_Error             ; ACK OK

       return

  if _CLOCK_STRETCH_CHECK
;*************************************************************************************************
;                    Fatal Error On I2C Bus
;
;  Slave pulling clock for too long or if SCL Line is stuck low.
;  This occurs if during Transmission, SCL is stuck low for period longer than appox 1mS
;   and RTCC times out ( appox 4096 cycles : 256 * 16 – prescaler of 16).
;
;*************************************************************************************************

Bus_Fatal_Error:

; diable RTCC Interrupt
;
       bcf     rtie                  ; disable RTCC interrupts, until next TXMT try

  RELEASE_BUS
;
;  Set the Bus_Status Bits appropriately
;
       bsf     Abort                 ; transmission was aborted
       bsf     Fatal_Error           ; FATAL Error occured
       bcf     Txmt_Progress         ; Transmission Is Not in Progress
       bcf     Txmt_Success          ; Transmission Unsuccesful
;
       call    TxmtStopBit           ; Try sending a STOP bit, may be not successful
;
  return
```

© 1994 Microchip Technology Inc.

```
;
;*****************************************************************************************************************
  endif


;*****************************************************************************************************************
;             General Purpose Delay Routines
;
;  Delay4uS    is wait loop for 4.0 uSec
;  Delay47uS   is wait loop for 4.7 uSec
;  Delay50uS   is wait loop for 5.0 uSec
;
;*****************************************************************************************************************
;

Delay50uSec:
  movlw ((_50uS_Delay-5)/3 + 1)
DlyK
  movwf DelayCount
  decfsz        DelayCount
  goto  $-1
  return
;
Delay47uSec:
  movlw ((_47uS_Delay-8)/3 + 1)
        goto    DlyK
;
Delay40uSec:
  movlw ((_40uS_Delay-8)/3 + 1)
  goto  DlyK
;
;*****************************************************************************************************************
```


© 1994 Microchip Technology Inc.

Appendix D - HIGH.ASM

```
;*****************************************************************************************************
;
;                                I2C Master : General Purpose Macros & Subroutines
;
;              High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
;
;*****************************************************************************************************

;*****************************************************************************************************
;
;                                I2C_TEST_DEVICE
; MACRO
;
;   If Slave Device is listening, then _SlaveActive bit is set, else is cleared
;
; Parameter :  NONE
;
;   Sequence Of Operations :
;     S-SlvAW-A-P
;   If A is +ve device is listening, else either busy, not present or error condition
;
;   This test may also be used to check for eample if a Serial EEPROM is in internal programming
;   mode
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;              mode addressing must be set
;*****************************************************************************************************
I2C_TEST_DEVICE         MACRO

            call    IsSlaveActive           ; TEMP ???? : Assembler Error with this MACRO

            ENDM
;
;            Test If A Device of SlaveAddr Is Present on Bus
;
;  The Slave Address Is put on the bus and if ACK it is present, if NACK not present
;  or may be device is not responding. The presense can be checked constantly by a master
;  (for ex. the Operating System on an Access.Bus may constantly issue this command)
;
;  Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
;   Set  _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
;
;   Returns  1 in _SlaveActive Bit if slave is responding else a 0
```


© 1994 Microchip Technology Inc.

```
;
;
IsSlaveActive:
        bcf     Slave_RW                ; set for write operation
        call    TxmtStartBit            ; send START bit
        call    Txmt_Slave_Addr         ; if successful, then _Txmt_Success bit is set
;
        bcf     SlaveActive
        btfss   ACK_Error               ; skip if NACK, device is not present or not responding
        bsf     SlaveActive             ; ACK received, device present & listening
        call    TxmtStopBit
        return
;
;*****************************************************************************************************
;                               I2C_WRITE
;
;  A basic macro for writing a block of data to a slave
;
;  Parameters :
;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_
;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
;
;   Sequence :
;               S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
;
;  If an error occurs then the routine simply returns and user should check for
;       flags in Bus_Status Reg (for eg. _Txmt_Success flag)
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;               mode addressing must be set
;*****************************************************************************************************

I2C_WR          MACRO   _BYTES_, _SourcePointer_

        movlw   BYTES_
        movwf   tempCount
        movlw   SourcePointer_
        movwf   fsr

        call    i2c_block_write
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission

        ENDM

_i2c_block_write:
        call    TxmtStartBit            ; send START bit
        bcf     Slave_RW                ; set for write operation
```


© 1994 Microchip Technology Inc.

```
        call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
;
_block_wr1_loop:
        btfss           Txmt_Success
        return
        movf            indf,w
        movwf           DataByte                ; start from the first byte starting at _DataPointer_
        incf            fsr
        call            SendData                ; send next byte, bus is our's !
        decfsz          tempCount
        goto    _       block_wr1_loop          ; loop until desired bytes of data transmitted to slave
        return
;
;*****************************************************************************************************
;*****************************************************************************************************
;                               I2C_WRITE_SUB
;
;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
;  to a slave device.
;               Eg. : A serial EEPROM would need an address of memory location for Random Writes
;
;  Parameters :
;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
;               _Sub_Address_           Sub-address of Slave (constant)
;
;   Sequence :
;               S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P
;
;  If an error occurs then the routine simply returns and user should check for
;       flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
;       Returns :       WREG = 1 on success, else WREG = 0
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;               mode addressing must be set
;
;  COMMENTS :
;               I2C_WR may prove to be more efficient than this macro in most situations
;               Advantages will be found for Random Address Block Writes for Slaves with
;               Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMS)
;
;*****************************************************************************************************

I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

        movlw           (_BYTES_ + 1)
        movwf           tempCount
```

© 1994 Microchip Technology Inc.

```
        movlw   (_SourcePointer_ - 1)
        movwf   fsr

        movf    indf,w
        movwf   StoreTemp_1             ; temporarily store contents of (_SourcePointer_ -1)
        movlw   Sub_Address_
        movwf   indf                    ; store temporarily the sub-address at (_SourcePointer_ -1)

        call    i2c_block_write         ; write _BYTES_+1 block of data

        movf    StoreTemp_1,w
        movwf   (_SourcePointer_- 1)    ; restore contents of (_SourcePointer_ - 1)

        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission

        ENDM

;*****************************************************************************************************
;                               I2C_WR_SUB_SWINC
;
;  Parameters :
;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
;               _Sub_Address_           Sub-address of Slave (constant)
;
;   Sequence :
;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
;                     and so on until #of Bytes
;
;  If an error occurs then the routine simply returns and user should check for
;       flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
;       Returns :       WREG = 1 on success, else WREG = 0
;
;  COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
;            Some I2C devices addresed with a sub-address do not increment automatically
;       after an access of each byte. Thus a block of data sent must have a sub-address
;       followed by a data byte.
;
;*****************************************************************************************************

I2C_WR_SUB_SWINC        MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                        variable i              ; TEMP ???? : Assembler Does Not Support This

                        i = 0
```


© 1994 Microchip Technology Inc.

```
                .while (i < _BYTES_)
  movf          (_Source_Pointer_ + i),w
  movwf         SrcPtr
  movf          (_Sub_Address_ + i),w
  movwf         SubAddr
  call          i2c_byte_wr_sub                 ; write a byte of data at sub address

                              i++
                .endw

                ENDM

;
;
;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;
_i2c_byte_wr_sub:
        call    TxmtStartBit                    ; send START bit
        bcf     Slave_RW                        ; set for write operation
        call    Txmt_Slave_Addr                 ; if successful, then _Txmt_Success bit is set
        btfss   Txmt_Success
        goto    block_wr1_fail                  ; end
        movf    SubAddr,w
        movwf   DataByte                        ; start from the first byte starting at _DataPointer_
        call    SendData                        ; send next byte
        btfss   Txmt_Success
        goto    block_wr1_fail                  ; end
        movf    SrcPtr,w
        movwf   DataByte                        ; start from the first byte starting at _DataPointer_
        call    SendData                        ; send next byte
        btfss   Txmt_Success
        goto    block_wr1_fail                  ; failed, return 0 in WREG
        goto    block_wr1_pass                  ; successful, return 1 in WREG
;
; return back to called routine from either _block_wr1_pass or _block_wr1_fail
;
_block_wr1_fail:
        call    TxmtStopBit                     ; Issue a stop bit for slave to end transmission
        retlw   FALSE
_block_wr1_pass:
        call    TxmtStopBit                     ; Issue a stop bit for slave to end transmission
        retlw   TRUE
;
;*****************************************************************************************************
;
;                               I2C_WR_MEM_BYTE
```


© 1994 Microchip Technology Inc.

```
;
;  Some I2C devices like a EEPROM need to wait fo some time after every byte write
;  (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
;  but in addition adds a delay after each byte.
;          Some EERPOM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
;          this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
;          for a block of data and then insert a delay until the whole buffer is written.
;
;  Parameters :
;          _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
;          _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
;          _Sub_Address_           Sub-address of Slave (constant)
;
;   Sequence :
;          S-SlvAW-A-(SubA+0)-A-D[0]-A-P
;                  Delay 1 mSec            ; The user can chnage this value to desired delay
;          S-SlvAW-A-(SubA+1)-A-D[1]-A-P
;                  Delay 1 mSec
;                  and so on until #of Bytes
;
;*****************************************************************************************************

I2C_WR_BYTE_MEM     MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                    variable i                  ; TEMP ???? : Assembler Does Not Support This

                    i = 0

                    .while (i < _BYTES_)
  movf    (_Source_Pointer_ + i),w
  movwf   SrcPtr
  movf    (_Sub_Address_ + i),w
                    movwf   SubAddr
                    call    _i2c_byte_wr_sub    ; write a byte of data at sub address
                    call    Delay50uSec

                            i++
                     .endw


                ENDM

;*****************************************************************************************************
;                   I2C_WR_MEM_BUF
;
;   This Macro/Function writes #of _BYTES_ to an I2C memory device. However
;   some devices, esp. EEPROMs must wait while the device enters into programming
;   mode. But some devices have an onchip temp data hold buffer and is used to
;  store data before the device actually enters into programming mode.
```


© 1994 Microchip Technology Inc.

```
;       For example, the 24C04 series of Serial EEPROMs from Microchip
;      have an 8 byte data buffer. So one can send 8 bytes of data at a time
;      and then the device enters programming mode. The master can either wait
;      until a fixed time and then retry to program or can continiously poll
;      for ACK bit and then transmit the next Block of data for programming
;
;  Parameters :
;               BYTES_              # of bytes to write to memory
;               _       SourcePointer_          Pointer to the block of data
;               _       SubAddress_     Sub-address of the slave
;               _       Device__BUF_SIZE        The on chip buffer size of the i2c slave
;
;  Sequence of operations
;               I2C_SUB_WR operations are performed in loop and each time
;               data buffer of BUF_SIZE is output to the device. Then
;               the device is checked for busy and when not busy another
;               block of data is written
;
;*************************************************************************************************
I2C_WR_BUF_MEM  MACRO   _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
      variable i, j

      if ( !_BYTES_)
             exitm
      elif ( _BYTES_ <=  _Device_BUF_SIZE_)

      I2C_WR_SUB      _BYTES_, _SourcePointer_, _SubAddress_

                    exitm

             else

             i = 0
             j = (_BYTES_ / _Device_BUF_SIZE_)
             .while (i < j)

                    I2C_WR_SUB      _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ +
 i*_Device_BUF_SIZE_)
                    call  IsSlaveActive
                    btfss   _SlaveActive
                    goto    $-2

                       i++
```


© 1994 Microchip Technology Inc.

```
                .endw

                j = (_BYTES_ - i*_Device_BUF_SIZE_)
                        if (j)
                        I2C_WR_SUB      j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ + i*_Device_BUF_SIZE_)
                         endif
        endif

        ENDM


;*****************************************************************************************************************
;
;                               I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
;   Parameters :
;               _BYTES_         :  constant : #of bytes to receive
;               _DestPointer_   :  destination pointer of RAM (File Registers)
;
;   Sequence :
;               S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;   If last byte, then Master will NOT Acknowledge (send NACK)
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;               mode addressing must be set
;
;*****************************************************************************************************************

I2C_READ        MACRO   _BYTES_, _DestPointer_


        movlw   (_BYTES_ -1)
        movwf   tempCount               ; -1 because, the last byte is used out of loop
        movlw   DestPointer_
        movwf   fsr                     ; FIFO destination address pointer

        call    i2c_block_read

        ENDM

_i2c_block_read:
        call    TxmtStartBit            ; send START bit
        bsf     Slave_RW                ; set for read operation
        bcf     Last_Byte_Rcv           ; not a last byte to rcv
```


© 1994 Microchip Technology Inc.

```
        call    Txmt_Slave_Addr         ; if successful, then _Txmt_Success bit is set
        btfsc   Txmt_Success
        goto    block_rd1_loop          ; end
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission
        retlw   FALSE                   ; Error : may be device not responding
;
_block_rd1_loop:
        call    GetData
        movf    DataByte,w
        movwf   indf                    ; start receiving data, starting at Destination Pointer
        incf    fsr
        decfsz  tempCount
        goto    block_rd1_loop          ; loop until desired bytes of data transmitted to slave
        bsf     Last_Byte_Rcv           ; last byte to rcv, so send NACK
        call    GetData
        movf    DataByte,w
        movwf   indf
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission
        retlw   TRUE

;*****************************************************************************************************
;
;                               I2C_READ_SUB
;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
;  Between the sub-addrers write & the following reads, a STOP condition is not issued and
;  a "REPEATED START" condition is used so that an other master will not take over the bus,
;  and also that no other master will overwrite the sub-address of the same salve.
;
;   This function is very commonly used in accessing Random/Sequential reads from a
;     memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
;
;  Parameters :
;               _BYTES_         # of bytes to read
;               _DestPointer_   The destination pointer of data to be received.
;               _BubAddress_    The sub-address of the slave
;
;  Sequence :
;      S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;
;*****************************************************************************************************

I2C_READ_SUB    MACRO   _BYTES_, _DestPointer_, _SubAddress_

        bcf     Slave_RW                ; set for write operation
        call    TxmtStartBit            ; send START bit
        call    Txmt_Slave_Addr         ; if successful, then _Txmt_Success bit is set
```


© 1994 Microchip Technology Inc.

```
        movlw   SubAddress_
        movwf   DataByte            ; START address of EEPROM(slave 1)
        call    SendData            ; write sub address
;
; do not send STOP after this, use REPEATED START condition
;
        I2C_READ _BYTES_, _DestPointer_


        ENDM


;*****************************************************************************************************
;                               I2C_READ_STATUS
;
;  This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
;  send a status byte upon reception of a control byte
;               This is basically same as I2C_READ MACRO for reading a single byte
;
;   For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
;       data at the current address location
;
;  On success WREG = 1 else = 0
;
;*****************************************************************************************************

I2C_READ_STATUS MACRO   _DestPointer_

        call    TxmtStartBit        ; send START bit
        bsf     Slave_RW            ; set for read operation
        call    Txmt_Slave_Addr     ; if successful, then _Txmt_Success bit is set
        btfsc   Txmt_Success
        goto    byte_rd1_loop       ; read a byte
        call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
        retlw   FALSE               ; Error : may be device not responding
_byte_rd1_loop:
        bsf     Last_Byte_Rcv       ; last byte to rcv, so send NACK
        call    GetData
        movf    DataByte,w
        movwf   DestPointer_
        call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
        btfss   Rcv_Success
        retlw   FALSE
        retlw   TRUE
```


© 1994 Microchip Technology Inc.

```
        ENDM

;*****************************************************************************************************************
I2C_READ_BYTE   MACRO   _DestPointer_
        I2C_READ_STATUS MACRO   _DestPointer_
        ENDM


;*****************************************************************************************************************
;                               I2C_WR_SUB_WR
;
;  This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This
;  may be useful for devices which need 2 blocks of data in which the first block may be an
;  extended address of a slave device. For example, a large I2C memory device, or a teletext
;  device with an extended addressing scheme, may need multiple bytes of data in the 1st block
;  that represents the actual physical address and is followed by a 2nd block that actually
;  represents the data.
;
;  Parameters :
;
;               _       BYTES1_                 1st block #of bytes
;               _       SourcePointer1_         Start Pointer of the 1st block
;               _       SubAddress_             Sub-Address of slave
;               _       BYTES2_                 2st block #of bytes
;               _       SourcePointer2_         Start Pointer of the 2nd block
;
;  Sequence :
;           S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
;
;  Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
;               a STOP bit is not sent (bus is not given up) in between
;               the two I2C_WR_SUB
;
;  Check Txmt_Success flag for any transmission errors
;
;*****************************************************************************************************************
I2C_WR_SUB_WR   MACRO   _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_

        movlw   (_COUNT1_ + 1)
        movwf   tempCount
        movlw   (_SourcePointer1_ - 1)
        movwf   fsr

        movf    indf,w
```


© 1994 Microchip Technology Inc.

```
        movwf   StoreTemp_1             ; temporarily store contents of (_SourcePointer_ -1)
        movlw   Sub_Address_
        movwf   indf                    ; store temporarily the sub-address at (_SourcePointer_ -1)
        call    i2c_block_write         ; write _BYTES_+1 block of data
;
        movf    StoreTemp_1,w
        movwf   (_SourcePointer1_- 1)  ; restore contents of (_SourcePointer_ - 1)
; Block 1 write over
; Send Block 2
        movlw   COUNT2_
        movwf   tempCount
        movlw   SourcePointer2_
        movwf   fsr
        call    block_wr1_loop
;
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission

        ENDM


;*************************************************************************************************
;                       I2C_WR_SUB_RD
;
;  This macro writes a block of data from SourcePointer of length _COUNT1_ to a slave
;       at sub-address and then Reads a block of Data of length _COUNT2_ to destination
;       address pointer
;
;  Message Structure :
;               S-SlvW-A-SubA-A-D1[0]-A-.....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P
;
;  Parameters :
;       _COUNT1_            Length Of Source Buffer
;       _SourcePointer_     Source Pointer Address
;       _Sub_Address_       The Sub Address Of the slave
;       _COUNT2_            The length of Destination Buffer
;       _DestPointer_       The start address of Destination Pointer
;
;*************************************************************************************************

I2C_WR_SUB_RD   MACRO   _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

        movlw   (_COUNT1_ + 1)
        movwf   tempCount
        movlw   (_SourcePointer_ - 1)
        movwf   fsr

        movf    indf,w
```


© 1994 Microchip Technology Inc.

```
        movwf   StoreTemp_1             ; temporarily store contents of (_SourcePointer_ -1)
        movlw   Sub_Address_
        movwf   indf                    ; store temporarily the sub-address at (_SourcePointer_ -1)
        call    i2c_block_write         ; write _BYTES_+1 block of data
;
        movf    StoreTemp_1,w
        movwf   (_SourcePointer1_- 1)  ; restore contents of (_SourcePointer_ - 1)
;
; Without sending a STOP bit, read a block of data by using a REPEATED
;  Start Condition
;
        I2C_READ        _COUNT2_, _DestPointer_

        ENDM


;*****************************************************************************************************
;
;                               I2C_WR_COM_WR
;
;  This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
;  the bus after sending the first block.
;               For example, this kind of transaction is used in an LCD driver where a
;       a block of control & addresss info is needed and then another block of actual data
;       to be displayed is needed.
;
;
;  Message Structure :
;               S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-......-A-D2[M-1]-A-P
;  NOTE : This message is same as calling two I2C_WR Macros, except that
;       the bus is not given up between the sending of 2 blocks (this is
;       done by not sending a STOP bit inbetween)
;
;  Parameters :
;       _COUNT1_                Length Of Source Buffer #1
;       _SourcePointer1_        Source Pointer Address of 1st buffer
;       _COUNT2_                The length of Destination Buffer
;       _SourcePointer2_        Source Pointer Address of 2nd Buffer
;
;*****************************************************************************************************
I2C_WR_COM_WR   MACRO   _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_
        movlw   COUNT1_
        movwf   tempCount
        movlw   SourcePointer1_
        movwf   fsr
        call    i2c_block_write
;
```


© 1994 Microchip Technology Inc.

```
; First block sent, now send 2nd block of data
;
        movlw   COUNT2_
        movwf   tempCount
        movlw   SourcePointer2__
        movwf   fsr
        call    block_wr1_loop
;
        call    TxmtStopBit             ; End of Double buffer txmt

        ENDM
;*****************************************************************************************************
;                               INCLUDE I2C Low Level Routines Here
;*****************************************************************************************************

        include "i2c_low.asm"
```


© 1994 Microchip Technology Inc.

Appendix E - I²C test.lst

```
MPASM B0.24                                                    PAGE  1
"I2C Master Mode Implemetation"
"Rev 0.1        :  01 Mar 1993"

                        Title           "I2C Master Mode Implemetation"
                        SubTitle        "Rev 0.1        :  01 Mar 1993"

        ;*****************************************************************************
        ;
        ;           Software Implementation Of I2C Master Mode
        ;
        ;   * Master Transmitter & Master Receiver Implemented in software
        ;   * Slave Mode implemented in hardware
        ;
        ;   *   Refer to Signetics/Philips I2C-Bus Specification
        ;
        ;   The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
        ;
        ;   RB1 is SDA          (Any I/O Pin May Be used instead)
        ;   RB0/INT is SCL      (Any I/O Pin May Be used instead)
        ;
        ;
        ;*****************************************************************************************************************

                                Processor       16C71
                                Radix   DEC

00F4 2400           _ClkIn          equ     16000000        ; Input Clock Frequency Of PIC16C71

                    include     "d:\pictools\16Cxx.h"
;
00A0                #define _Slave_1_Addr   0xA0                    ; Serial EEPROM #1
00AC                #define _Slave_2_Addr   0xAC                    ; Serial EEPROM #2
00D6                #define _Slave_3_Addr   0xD6                    ; Slave PIC16CXX
0046                #define _ENABLE_BUS_FREE_TIME   TRUE
0047                #define _CLOCK_STRETCH_CHECK    TRUE
0048                #define _INCLUDE_HIGH_LEVEL_I2C TRUE

                        include     "i2c.h"
                    ;*****************************************************************************************************
                    ;               I2C Bus Header File
                    ;*****************************************************************************************************

003D 0900           _ClkOut         equ     (_ClkIn >> 2)
```


© 1994 Microchip Technology Inc.

```
                    ;
                    ; Compute the delay constants for setup & hold times
                    ;
0010                  _40uS_Delay     set     (_ClkOut/250000)
0012                  _47uS_Delay     set     (_ClkOut/212766)
0014                  _50uS_Delay     set     (_ClkOut/200000)

0049                #define _OPTION_INIT    (0xC0 | 0x03)           ; Prescaler to RTCC for Appox 1 mSec timeout
                    ;
004A                #define _SCL      _     portb,0
004B                #define _SDA      _     portb,1

004C                #define _SCL_TRIS       trisb,0
004D                #define _SDA_TRIS       trisb,1

0000                #define _WRITE_         0
0001                #define _READ_          1
                    ;       Register File Variables

                            CBLOCK  0x0C
000C 0001                       SlaveAddr       ; Slave Addr must be loader into this reg
000D 0001                       SlaveAddrHi     ; for 10 bit addressing mode
000E 0001                       DataByte        ; load this reg with the data to be transmitted
000F 0001                       BitCount        ; The bit number (0:7) transmitted or received
0010 0001                       Bus_Status      ; Status Reg of I2C Bus for both TXMT & RCVE
0011 0001                       Bus_Control     ; control Register of I2C Bus
0012 0001                       DelayCount
0013 0001                       DataByteCopy    ; copy of DataByte for Left Shifts (destructive)
0014 0000
0014 0001                       SubAddr         ; sub-address of slave (used in I2C_HIGH.ASM)
0015 0001                       SrcPtr          ; source pointer for data to be transmitted
0016 0000
0016 0001                       tempCount       ; a temp variable for scratch RAM
0017 0001                       StoreTemp_1     ; a temp variable for scratch RAM, do not disturb contents
0018 0000
0018 0001                       End_I2C_Ram     ; unused, only for ref of end of RAM allocation
                                ENDC

                            ;*****************************************************************************
                            ;           I2C Bus Status Reg Bit Definitions
                            ;*****************************************************************************

004E                #define _Bus_Busy           Bus_Status,0
004F                #define _Abort              Bus_Status,1
0050                #define _Txmt_Progress      Bus_Status,2
0051                #define _Rcv_Progress       Bus_Status,3
0052                #define _Txmt_Success       Bus_Status,4
0053                #define _Rcv_Success        Bus_Status,5
0054                #define _Fatal_Error        Bus_Status,6
```


© 1994 Microchip Technology Inc.

```
0055                    #define _ACK_Error              Bus_Status,7

                        ;*****************************************************************************
                        ;       I2C Bus Control Register
                        ;*****************************************************************************************************
0056                    #define _10BitAddr          Bus_Control,0
0057                    #define _Slave_RW           Bus_Control,1
0058                    #define _Last_Byte_Rcv      Bus_Control,2
0059                    #define _SlaveActive        Bus_Control,6
005A                    #define _TIME_OUT_          Bus_Control,7


                        ;*****************************************************************************************************
                        ;           General Purpose Macros
                        ;*****************************************************************************************************

                        RELEASE_BUS     MACRO
                              bsf _rp0          ; select page 1
                              bsf _SDA          ; tristate SDA
                              bsf _SCL          ; tristate SCL
                              bcf _Bus_Busy     ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                           ENDM

                        ;*****************************************************************************************************
                        A MACRO To Load 8 OR 10 Bit Address To The Address Registers
                        ;
                        ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s)
                        ;  depending on 8 or 10 bit addressing modes
                        ;*****************************************************************************************************

                        LOAD_ADDR_10    MACRO   SLAVE_ADDRESS

                              bsf _   10BitAddr                                 ; Slave has 10 bit address
                              movlw   (SLAVE_ADDRESS & 0xff)
                              movwf   SlaveAddr                                 ; load low byte of address
                              movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)    ; 10 bit addr is 11110XX0
                              movwf   SlaveAddr+1                               ; hi order  address

                           ENDM
                        LOAD_ADDR_8     MACRO   SLAVE_ADDRESS

                              bcf _   10BitAddr                                 ; Set for 8 Bit Address Mode
                              movlw   (SLAVE_ADDRESS & 0xff)
                              movwf   SlaveAddr
                           ENDM


                        CBLOCK  _End_I2C_Ram
0018 0001                     SaveStatus                                        ; copy of STATUS Reg
0019 0001                     SaveWReg                                          ; copy of WREG
```

© 1994 Microchip Technology Inc.

```
001A 0001                       byteCount
001B 0001                       HoldData
                              ENDC

                              CBLOCK  0x20
0020 0001                             DataBegin         ; Data to be read or written is stored here
                              ENDC


                               ORG     0x00
0000 2956                      goto    Start
                    ;
                               ORG     0x04
                    ;*****************************************************************************************************
                    ;              Interrupt Service Routine
                    ;
                    ;   For I2C routines, only RTCC interrupt is used
                    ; RTCC Interrupts enabled only if Clock Stretching is Used
                    ; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
                    ; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
                    ; The user may choose to retry the operation later again
                    ;
                    ;*****************************************************************************************************

                    Interrupt:
                    ;
                    ; Save Interrupt Status (WREG & STATUS regs)
                    ;
0004 0099                           movwf   SaveWReg            ; Save WREG
0005 0E03                           swapf   status,w            ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098                           movwf   SaveStatus          ; Save STATUS Reg if _CLOCK_STRETCH_CHECK RTCC Interrupts enabled only
                                                                ;    if Clock Stretching is Used
0007 1D0B                           btfss   rtif
0008 280B                           goto    MayBeOtherInt       ; other Interrupts
0009 1791                           bsf     TIME_OUT_           ; MUST set this Flag, can take other desired actions here
000A 110B                           bcf     rtif
                                    endif
                    ;
                    ;  Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
                    ;
                    MayBeOtherInt:
000B 0000                           NOP
                    ;
                    RestoreIntStatus:                       ; Restore Interrupt Status
000C 0E18                           swapf   SaveStatus,w
000D 0083                           movwf   status          ; restore STATUS Reg
000E 0E99                           swapf   SaveWReg
```


© 1994 Microchip Technology Inc.

```
000F 0E19                swapf   SaveWReg,w            ; restore WREG
0010 0009                retfie
                         ;
                         ;*****************************************************************************************************
                         ;            Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
                         include    "i2c_high.asm"
                         ;*****************************************************************************************************
                         ;
                         ;     I2C Master : General Purpose Macros & Subroutines
                         ;
                         ;     High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
                         ;
                         ;*****************************************************************************************************


                         ;*****************************************************************************************************
                         ;
                         ;                              I2C_TEST_DEVICE
                         ; MACRO
                         ;
                         ;   If Slave Device is listening, then _SlaveActive bit is set, else is cleared
                         ;
                         ;  Parameter :  NONE
                         ;
                         ;  Sequence Of Operations :
                         ;       S-SlvAW-A-P
                         ;  If A is +ve device is listening, else either busy, not present or error condition
                         ;
                         ;  This test may also be used to check for eample if a Serial EEPROM is in internal programming mode
                         ;
                         ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers,
                         ;           and 10 or 8 bit mode addressing must be set
                         ;*****************************************************************************************
                            I2C_TEST_DEVICE         MACRO

                            call    IsSlaveActive           ; TEMP ???? : Assembler Error with this MACRO
                           ENDM
                         ;
                         ;
                         ;          Test If A Device of SlaveAddr Is Present on Bus
                         ;
                         ;  The Slave Address Is put on the bus and if ACK it is present, if NACK not
                         ;  present or may be device is not responding. The presense can be checked
                         ;  constantly by a master(for ex. the Operating System on an Access. Bus may
                         ;  constantly issue this command)
                         ;   Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
                         ;   Set  _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
```

© 1994 Microchip Technology Inc.

```
                        ;
                        ;   Returns  1 in _SlaveActive Bit if slave is responding else a 0
                        ;
                        ;
                        IsSlaveActive:
0011 1091                       bcf     Slave_RW            ; set for write operation
0012 2057                       call    TxmtStartBit                ; send START bit
0013 206B                       call    Txmt_Slave_Addr             ; if successful, then _Txmt_Success bit is set
                        ;
0014 1311                       bcf     SlaveActive
0015 1F90                       btfss   ACK_Error           ; skip if NACK, device is not present or not responding
0016 1711                       bsf     SlaveActive         ; ACK received, device present & listening
0017 205F                       call    TxmtStopBit
0018 0008                       return
                        ;
                        ;*****************************************************************************************************
                        ;                                       I2C_WRITE
                        ;
                        ;  A basic macro for writing a block of data to a slave
                        ;
                        ;  Parameters :
                        ;               _BYTES              #of bytes starting from RAM pointer _SourcePointer_
                        ;   SourcePointer_                  Data Start Buffer pointer in RAM (file Registers)
                        ;
                        ;   Sequence :
                        ;               S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
                        ;
                        ;  If an error occurs then the routine simply returns and user should check
                        ;   for flags in Bus_Status Reg (for eg. _Txmt_Success flag)
                        ;
                        ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers,
                        ;  and 10  or 8 bit mode addressing must be set
                        ;*****************************************************************************************************

                        I2C_WR          MACRO   _BYTES_, _SourcePointer_

                              movlw       BYTES_
                              movwf       tempCount
                              movlw       _SourcePointer_
                              movwf       _fsr

                              call        _i2c_block_write
                              call        TxmtStopBit                 ; Issue a stop bit for slave to end transmission

                           ENDM

                          _i2c_block_write:
0019 2057                       call      TxmtStartBit              ; send START bit
```


© 1994 Microchip Technology Inc.

```
001A 1091               bcf     Slave_RW                ; set for write operation
001B 206B               call    Txmt_Slave_Addr         ; if successful, then _Txmt_Success bit is set
                    ;
                    _block_wr1_loop:
001C 1E10               btfss   Txmt_Success
001D 0008               return
001E 0800               movf    indf,w
001F 008E               movwf   DataByte                ; start from the first byte starting at DataPointer_
0020 0A84               incf    fsr
0021 2095               call    SendData                ; send next byte, bus is our's !
0022 0B96               decfsz  tempCount
0023 281C               goto    block_wr1_loop          ; loop until desired bytes of data transmitted to slave
0024 0008               return
                ;
                ;*****************************************************************************************************
                ;*****************************************************************************************************
                ;                               I2C_WRITE_SUB
                ;
                ;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address  to a slave device.
                ;  Eg. : A serial EEPROM would need an address of memory location for Random Writes
                ;  Parameters :
                ;  BYTES        #of bytes starting from RAM pointer _SourcePointer_ (constant)
                ;  SourcePointer_          Data Start Buffer pointer in RAM (file Registers)
                ;  Sub_Address_            Sub-address of Slave (constant)
                ;
                ;   Sequence :
                ;                S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P
                ;
                ;  If an error occurs then the routine simply returns and user should check
                ;        for flags in Bus_Status Reg (for eg. _Txmt_Success flag
                ;
                ;       Returns :       WREG = 1 on success, else WREG = 0
                ;
                ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers,
                ;         and 10 or 8 bit mode addressing must be set
                ;
                ;  COMMENTS :
                ;  I2C_WR may prove to be more efficient than this macro in most situations
                ;  Advantages will be found for Random Address Block Writes for Slaves with
                ;  Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMS)
                ;
                ;*****************************************************************************************************

                I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                    movlw   (_BYTES_ + 1)
                    movwf   tempCount
```


© 1994 Microchip Technology Inc.

```
        movlw   (_SourcePointer_ - 1)
        movwf   fsr
        movf    indf,w
        movwf   StoreTemp_1             ; temporarily store contents of (_SourcePointer_ -1)
        movlw   Sub_Address_
        movwf   indf                    ; store temporarily the sub-address at (_SourcePointer_ -1)

        call    i2c_block_write         ; write _BYTES_+1 block of data
        movf    StoreTemp_1,w
        movwf   (_SourcePointer_ - 1)   ; restore contents of (_SourcePointer_ - 1)
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission
    ENDM

;*****************************************************************************************************
;                       I2C_WR_SUB_SWINC
;
;  Parameters :
;      BYTES   #of bytes starting from RAM pointer _SourcePointer_ (constant)
;      SourcePointer  Data Start Buffer pointer in RAM (file Registers)
;      Sub_Address    Sub-address of Slave (constant)
;
;   Sequence :
;              S-SlvAW-A-(SubA+0)-A-D[0]-A-P
;              S-SlvAW-A-(SubA+1)-A-D[1]-A-P
;                  and so on until #of Bytes
;
;  If an error occurs then the routine simply returns and user should check
;      for  flags in Bus_Status Reg (for eg. _Txmt_Success flag
;
;      Returns :       WREG = 1 on success, else WREG = 0
;
;  COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
;      Some I2C devices addresed with a sub-address do not increment
;      automatically after an access of each byte. Thus a block of data
;      sent must have a sub-address followed by a data byte.
;
;*****************************************************************************************************

 I2C_WR_SUB_SWINC       MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

    variable i                          ; TEMP ???? : Assembler Does Not Support This

                i = 0

            .while (i < _BYTES_)
            movf    (_Source_Pointer_ + i),w
```


© 1994 Microchip Technology Inc.

```
                        movwf   SrcPtr
                        movf    (_Sub_Address_ + i),w
                        movwf   SubAddr
                        call    i2c_byte_wr_sub     ; write a byte of data at sub address
                                                    i++
                        .endw
                    ENDM

                ;
                ;
                ;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
                ;

                i2c_byte_wr_sub:
0025 2057       call    TxmtStartBit        ; send START bit
0026 1091       bcf     Slave_RW            ; set for write operation
0027 206B       call    Txmt_Slave_Addr     ; if successful, then _Txmt_Success bit is set
0028 1E10       btfss   Txmt_Success
0029 2835       goto    block_wr1_fail      ; end
002A 0814       movf    SubAddr,w
002B 008E       movwf   DataByte            ; start from the first byte starting at DataPointer_
002C 2095       call    SendData            ; send next byte
002D 1E10       btfss   Txmt_Success
002E 2835       goto    block_wr1_fail      ; end
002F 0815       movf    SrcPtr,w
0030 008E       movwf   DataByte            ; start from the first byte starting at DataPointer_
0031 2095       call    SendData            ; send next byte
0032 1E10       btfss   Txmt_Success
0033 2835       goto    block_wr1_fail      ; failed, return 0 in WREG
0034 2837       goto    block_wr1_pass      ; successful, return 1 in WREG
                ;
                ; return back to called routine from either _block_wr1_pass or block_wr1_fail
                block_wr1_fail:
0035 205F       call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
0036 3400       retlw   FALSE
                block_wr1_pass:
0037 205F       call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
0038 3401       retlw   TRUE
                ;

;*****************************************************************************************************

                ;
                ;                           I2C_WR_MEM_BYTE
                ;
```

```
;  Some I2C devices like a EEPROM need to wait fo some time after every byte  write (when entered into
;  internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC, but in addition adds a delay
;  after each byte.   Some EERPOM memories (like Microchip's 24Cxx Series have on-chip data buffer),
;  and hence this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB for a
;  block of data and then insert a delay until the whole buffer is written.
;  Parameters :
;       BYTES_  #of bytes starting from RAM pointer _SourcePointer_ (constant)
;       SourcePointer_      Data Start Buffer pointer in RAM (file Registers)
;       Sub_Address_        Sub-address of Slave (constant)
;
;   Sequence :
;                S-SlvAW-A-(SubA+0)-A-D[0]-A-P
;        Delay 1 mSec       ; The user can chnage this value to desired delay
;                S-SlvAW-A-(SubA+1)-A-D[1]-A-P
;                       Delay 1 mSec
;                       and so on until #of Bytes
;
;*************************************************************************************************

I2C_WR_BYTE_MEM         MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

        variable i          ; TEMP ???? : Assembler Does Not Support This

                            i = 0

                            .while (i < _BYTES_)
        movf    (_Source_Pointer_ + i),w
        movwf   SrcPtr
        movf    (_Sub_Address_ + i),w
        movwf   SubAddr
        call    i2c_byte_wr_sub             ; write a byte of data at sub address
        call    Delay50uSec
                                    i++
                            .endw


        ENDM
;*************************************************************************************************
;                       I2C_WR_MEM_BUF
;
;  This Macro/Function writes #of _BYTES_ to an I2C memory device. However some devices, esp. EEPROMs must
;  wait while the device enters into programming mode. But some devices have an onchip temp data hold buffer
;  and is used to store data before the device actually enters into programming mode.  For example, the 24C04
;  series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a
;  time and then the device enters programming mode. The master can either wait until a fixed time and then
;  retry to program or can continiously poll for ACK bit and then transmit the next Block of data for programming
```


© 1994 Microchip Technology Inc.

```
;
;  Parameters :
;      BYTES_          # of bytes to write to memory
;      SourcePointer_  Pointer to the block of data
;      SubAddress_     Sub-address of the slave
;      Device_BUF_SIZE_  The on chip buffer size of the i2c slave
;
;  Sequence of operations
;   I2C_SUB_WR operations are performed in loop and each time
;   data buffer of BUF_SIZE is output to the device. Then
;   the device is checked for busy and when not busy another
;   block of data is written
;
;
;*****************************************************************************

I2C_WR_BUF_MEM  MACRO   _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_

      variable i, j

      if ( !_BYTES_)

          exitm

      elif ( _BYTES_ <=  _Device_BUF_SIZE_)

      I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_

                              exitm

                      else

      i = 0
      j = (_BYTES_ / _Device_BUF_SIZE_)
      .while (i < j)

          I2C_WR_SUB      _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_),
          (_SubAddress_ + i*_Device_BUF_SIZE_)

          call    IsSlaveActive
          btfss   _SlaveActive
          goto    $-2

                                  i++
      .endw

          j = (_BYTES_ - i*_Device_BUF_SIZE_)

                                  if (j)
```


© 1994 Microchip Technology Inc.

```
I2C_WR_SUB j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ + i*_Device_BUF_SIZE_)

        endif
    endif
ENDM
;*****************************************************************************************************************
;
;                                   I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
;   Parameters :
;                BYTES_          :  constant : #of bytes to receive
;              _ DestPointer_    :  destination pointer of RAM (File Registers)
;
;   Sequence :
;              S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;   If last byte, then Master will NOT Acknowledge (send NACK)
;
;  NOTE :  The address of the slave must be loaded into SlaveAddress Registers, and
;          10 or 8 bit mode addressing must be set
;
;*****************************************************************************************************************

I2C_READ        MACRO   _BYTES_, _DestPointer_

    movlw   (_BYTES_ -1)
    movwf    tempCount          ; -1 because, the last byte is used out of loop
    movlw   DestPointer_
    movwf   fsr                 ; FIFO destination address pointer
    call    i2c_block_read
    ENDM

    _i2c_block_read:
0039 2057       call    TxmtStartBit        ; send START bit
003A 1491       bsf     Slave_RW            ; set for read operation
003B 1111       bcf     Last_Byte_Rcv       ; not a last byte to rcv
003C 206B       call    Txmt_Slave_Addr     ; if successful, then _Txmt_Success bit is set
003D 1A10       btfsc   Txmt_Success
003E 2841       goto    block_rd1_loop      ; end
003F 205F       call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
0040 3400       retlw   FALSE               ; Error : may be device not responding
                                            ;
    _block_rd1_loop:
0041 20CC       call    GetData
```


© 1994 Microchip Technology Inc.

```
0042 080E               movf    DataByte,w
0043 0080               movwf   indf                    ; start receiving data, starting at Destination Pointer

0044 0A84               incf    fsr
0045 0B96               decfsz  tempCount
0046 2841               goto    block_rd1_loop          ; loop until desired bytes of data transmitted to slave
0047 1511               bsf     Last_Byte_Rcv           ; last byte to rcv, so send NACK
0048 20CC               call    GetData
0049 080E               movf    DataByte,w
004A 0080               movwf   indf
004B 205F               call    TxmtStopBit             ; Issue a stop bit for slave to end transmission
004C 3401               retlw   TRUE
```

```
;*************************************************************************************************
;
;                                   I2C_READ_SUB
;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address Between the
;  sub-addrers write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is
;  used so that an other master will not take over the bus, and also that no other master will overwrite the sub-
;  address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a
;  memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
;
;  Parameters :
;      BYTES_          # of bytes to read
;      DestPointer_    The destination pointer of data to be received.
;      BubAddress_     The sub-address of the slave
;
;  Sequence :
;      S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;
;*************************************************************************************************

I2C_READ_SUB    MACRO   _BYTES_, _DestPointer_, _SubAddress_

    bcf     Slave_RW            ; set for write operation
    call    TxmtStartBit        ; send START bit
    call    Txmt_Slave_Addr     ; if successful, then _Txmt_Success bit is set

    movlw   SubAddress_
    movwf    DataByte           ; START address of EEPROM(slave 1)
    call     SendData           ; write sub address
;
; do not send STOP after this, use REPEATED START condition
;

I2C_READ _BYTES_, _DestPointer_
```

© 1994 Microchip Technology Inc.

```
    ENDM

;*****************************************************************************************************
;
;                                I2C_READ_STATUS
;
;  This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can send a status byte
;  upon reception of a control byte.    This is basically same as I2C_READ MACRO for reading a single byte.
;
;  For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory data at the
;  current address location
;  On success WREG = 1 else = 0
;
;*****************************************************************************************************

I2C_READ_STATUS MACRO   _DestPointer_


   call    TxmtStartBit          ; send START bit
   bsf     Slave_RW              ; set for read operation
   call    Txmt_Slave_Addr       ; if successful, then _Txmt_Success bit is set
   btfsc   Txmt_Success
   goto    byte_rd1_loop         ; read a byte
   call    TxmtStopBit           ; Issue a stop bit for slave to end transmission
   retlw   FALSE                 ; Error : may be device not responding
   _byte_rd1_loop:
   bsf     Last_Byte_Rcv         ; last byte to rcv, so send NACK
   call    GetData
   movf    DataByte,w
   movwf   DestPointer_
   call    TxmtStopBit           ; Issue a stop bit for slave to end transmission
   btfss   Rcv_Success
   retlw   FALSE
   retlw   TRUE
   ENDM

;*****************************************************************************************************

I2C_READ_BYTE   MACRO   _DestPointer_

I2C_READ_STATUS MACRO   _DestPointer_

   ENDM

;*****************************************************************************************************
```


© 1994 Microchip Technology Inc.

```
;
;                                   I2C_WR_SUB_WR
;
;  This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This may be useful for
;  devices which need 2 blocks of data in which the first block may be an extended address of a slave device.
;  For example, a large I2C memory device, or a teletext device with an extended addressing scheme, may need
;  multiple bytes of data in the 1st block  that represents the actual physical address and is followed by a
;  2nd block that actually represents the data.
;
;  Parameters :
;
;              BYTES1          1st block #of bytes
;              SourcePointer1  Start Pointer of the 1st block
;                 SubAddress_  Sub-Address of slave
;              BYTES2_         2st block #of bytes
;              SourcePointer2  Start Pointer of the 2nd block
;
;  Sequence :
;        S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
;
;  Note :  This MACRO is basically same as calling I2C_WR_SUB twice, but
;          a STOP bit is not sent (bus is not given up) in between
;          the two I2C_WR_SUB
;
;  Check Txmt_Success flag for any transmission errors
;
;*************************************************************************************************************

I2C_WR_SUB_WR   MACRO   _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, SourcePointer2_

   movlw    (_COUNT1_ + 1)
   movwf    tempCount
   movlw    (_SourcePointer1_ - 1)
   movwf    fsr
   movf     indf,w
   movwf    StoreTemp_1                    ; temporarily store contents of (_SourcePointer_ -1)
   movlw    _Sub_Address_
   movwf    _indf                          ; store temporarily the sub-address at (_SourcePointer_1)
   call     i2c_block_write                ; write _BYTES_+1 block of data
;
   movf     StoreTemp_1,w
   movwf    (_SourcePointer1_- 1) ; restore contents of (_SourcePointer_ - 1)
                  ; Block 1 write over
                  ; Send Block 2
   movlw    COUNT2_
   movwf    tempCount
   movlw    SourcePointer2_
   movwf    fsr
   call     block_wr1_loop
```


© 1994 Microchip Technology Inc.

```
    call    TxmtStopBit                 ; Issue a stop bit for slave to end transmission
    ENDM

;*****************************************************************************************************
;
;                       I2C_WR_SUB_RD
;
;  This macro writes a block of data from SourcePointer of length _COUNT1_ to a
;  slave at sub-address and then Reads a block of Data of length _COUNT2_ to
;  destination address pointer
;
;
;  Message Structure :
;  S-SlvW-A-SubA-A-D1[0]-A-....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P
;
;  Parameters :
;       _COUNT1_            Length Of Source Buffer
;       _SourcePointer_     Source Pointer Address
;       _Sub_Address_       The Sub Address Of the slave
;       _COUNT2_            The length of Destination Buffer
;       _DestPointer_       The start address of Destination Pointer
;
;*****************************************************************************************************

I2C_WR_SUB_RD   MACRO  _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

   movlw   (COUNT1_ + 1)
   movwf   tempCount
   movlw   (SourcePointer_ - 1)
   movwf   fsr
   movf    indf,w
   movwf   StoreTemp_1                 ; temporarily store contents of (_SourcePointer_ -1)
   movlw   Sub_Address_
   movwf   indf                        ; store temporarily the sub-address at (_SourcePointer_ -1)
   call    i2c_block_write             ; write _BYTES_+1 block of data
                                       ;
   movf    StoreTemp_1,w
   movwf   (SourcePointer1_- 1)        ; restore contents of (_SourcePointer_ - 1)
;
; Without sending a STOP bit, read a block of data by using a REPEATED
;  Start Condition
;
I2C_READ        _COUNT2_, _DestPointer_

   ENDM

;*****************************************************************************************************
;
```


© 1994 Microchip Technology Inc.

```
;                                   I2C_WR_COM_WR
;
;  This Macro write 2 blocks of data buffers to a slave in one message. This
;  way no need to give up the bus after sending the first block.
;     For example, this kind of transaction is used in an LCD driver where a
;  block of control & addresss info is needed and then another block of actual
;  data to be displayed is needed.
;
;
;  Message Structure :
;          S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.......-A-D2[M-1]-A-P
;  NOTE : This message is same as calling two I2C_WR Macros, except that
;         the bus is not given up between the sending of 2 blocks (this is
;         done by not sending a STOP bit inbetween)
;
;  Parameters :
;       _COUNT1_           Length Of Source Buffer #1
;       _SourcePointer1_   Source Pointer Address of 1st buffer
;       _COUNT2_           The length of Destination Buffer
;       _SourcePointer2_   Source Pointer Address of 2nd Buffer
;
;*****************************************************************************************************

I2C_WR_COM_WR   MACRO   _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_

    movlw       COUNT1_
    movwf       tempCount
    movlw       SourcePointer1_
    movwf       fsr
    call        i2c_block_write
;
; First block sent, now send 2nd block of data
;
    movlw   _   COUNT2_
    movwf       tempCount
    movlw   _   SourcePointer2__
    movwf   _   fsr
    call    _   block_wr1_loop
;
    call        TxmtStopBit                 ; End of Double buffer txmt

    ENDM

;*****************************************************************************************************
;                           INCLUDE I2C Low Level Routines Here
;*****************************************************************************************************

  include "i2c_low.asm"
;*****************************************************************************************************
```


© 1994 Microchip Technology Inc.

```
                    ;
                    ;             Low Level I2C Routines
                    ;
                    ;   Single Master Transmitter & Single Master Receiver Routines
                    ;   These routines can very easily be converted to Multi-Master System
                    ; when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
                    ;   detection is available.
                    ;
                    ;   The generic high level routines are given in I2C_HIGH.ASM
                    ;
                    ;*****************************************************************************************************

                    ;*****************************************************************************************************
                    ;             I2C Bus Initialization
                    ;
                    ;*****************************************************************************************************
                        InitI2CBus_Master:

004D 1283               bcf     _   rp0
004E 0806               movf    _   portb,w
004F 39FC               andlw       0xFC        ; do not use BSF & BCF on Port Pins
0050 0086               movwf       portb       ; set SDA & SCL to zero. From Now on, simply play with tris
                                                ; RELEASE_BUS
0051 1683
0051 1486
0052 1406
0054 0190               clrf    Bus_Status      ; reset status reg
0055 0191               clrf    Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008               return
                    ;
                    ;*****************************************************************************************************
                    ;                 Send Start Bit
                    ;
                    ;*****************************************************************************************************

                       TxmtStartBit:
0057 1683               bsf     rp0             ; select page 1
0058 1486               bsf     SDA             ; set SDA high
0059 1406               bsf     SCL             ; clock is high
                                                ;
                                                ; Setup time for a REPEATED START condition (4.7 uS)
                                                ;
005A 210D               call    Delay40uSec     ; only necesry for setup time
                                                ;
005B 1086               bcf     SDA             ; give a falling edge on SDA while clock is high
                                                ;
005C 210B               call    Delay47uSec     ; only necessary for START HOLD time
                                                ;
005D 1410               bsf     Bus_Busy        ; on a start condition bus is busy
```


© 1994 Microchip Technology Inc.

```
                    ;
005E 0008           return

                    ;*****************************************************************************************************
                    ;                       Send Stop Bit
                    ;
                    ;*****************************************************************************************************

                    TxmtStopBit:

005F 1683           bsf     rp0             ; select page 1
0060 1006           bcf     SCL
0061 1086           bcf     SDA             ; set SDA low
0062 1406           bsf     SCL             ; Clock is pulled up
0063 210D           call    Delay40uSec     ; Setup Time For STOP Condition
0064 1486           bsf     SDA             ; give a rising edge on SDA while CLOCK is high
                                            ;
                    if _ENABLE_BUS_FREE_TIME
                                            ; delay to make sure a START bit is not sent immediately after a STOP,
                                            ;   ensure BUS Free Time tBUF
                                            ;
0065 210B           call    Delay47uSec
                    endif
                                            ;
0066 1010           bcf     Bus_Busy        ; on a stop condition bus is considered Free
                                            ;
0067 0008           return

                    ;*****************************************************************************************************
                    ;                   Abort Transmission
                    ;
                    ;   Send STOP Bit & set Abort Flag
                    ;*****************************************************************************************************

                    AbortTransmission:

0068 205F           call    TxmtStopBit
0069 1490           bsf     Abort
006A 0008                       return

                    ;*****************************************************************************************************
                    ;               Transmit Address (1st Byte)& Put in Read/Write Operation
                    ;
                    ;  Transmits Slave Addr On the 1st byte and set LSB to R/W operation
                    ;  Slave Address must be loaded into SlaveAddr reg
                    ;  The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for
```

© 1994 Microchip Technology Inc.

```
                ;  Write & 1 for Read
                ;  On Success, return TRUE in WREG, else FALSE in WREG
                ;
                ;   If desired, the failure may tested by the bits in Bus Status Reg
                ;
                ;*****************************************************************************************************

                Txmt_Slave_Addr:
006B 1390           bcf         ACK_Error           ; reset Acknowledge error bit
006C 1C11           btfss       10BitAddr
006D 2886           goto        SevenBitAddr
                                                    ;
006E 1C91           btfss       Slave_RW
006F 287D           goto        TenBitAddrWR        ; For 10 Bit WR simply send 10 bit addr
                                                    ;
                                                    ; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then
                                                    ; Repeated Start and then Hi Byte Only for read opreation
                                                    ;
                TenBitAddrRd:

0070 1091           bcf         Slave_RW            ; temporarily set for WR operation
0071 207D           call        TenBitAddrWR
0072 1E10           btfss       Txmt_Success        ; skip if successful
0073 3400           retlw       FALSE
0074 2057           call        TxmtStartBit        ; send A REPEATED START condition
0075 1491           bsf         Slave_RW            ; For 10 bit slave Read

0076 080D           movf        SlaveAddr+1,W
0077 008E           movwf       DataByte
0078 140E           bsf         DataByte,LSB        ; Read Operation
0079 2095           call        SendData            ; send ONLY high byte of 10 bit addr slave
007A 288C           goto        AddrSendTest        ; 10 Bit Addr Send For Slave Read Over
                                                    ;
                                                    ; if successfully transmitted, expect an ACK bit
                                                    ;
007B 1E10           btfss       Txmt_Success        ; if not successful, generate STOP & abort transfer
007C 2890           goto        AddrSendFail
                                                    ;
                TenBitAddrWR:

007D 080D           movf    SlaveAddr+1,W
007E 008E           movwf   DataByte
007F 100E           bcf     DataByte,LSB            ; WR Operation
                                                    ;
                                                    ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW
                                                    ; Enabled) then save RETURN Address Pointer
                                                    ;
```


© 1994 Microchip Technology Inc.

```
0080 2095          call    SendData                ; send high byte of 10 bit addr slave
                                                   ;
                                                   ; if successfully transmitted, expect an ACK bit
                                                   ;
0081 1E10          btfss   Txmt_Success            ; if not successful, generate STOP & abort transfer
0082 2890          goto    AddrSendFail
                                                   ;
0083 080C          movf    SlaveAddr,W
0084 008E          movwf   DataByte                ; load addr to DatByte for transmission
0085 288B          goto    EndTxmtAddr

                   SevenBitAddr:
0086 080C          movf    SlaveAddr,W
0087 008E          movwf   DataByte                ; load addr to DatByte for transmission
0088 100E          bcf     DataByte,LSB
0089 1891          btfsc   Slave_RW                ; if skip then write operation
008A 140E          bsf     DataByte,LSB            ; Read Operation

                   EndTxmtAddr:
008B 2095          call    SendData                ; send 8 bits of address, bus is our's
                                                   ;
                                                   ; if successfully transmitted, expect an ACK bit
                                                   ;
                   _AddrSendTest:
008C 1E10          btfss   Txmt_Success            ; skip if successful
008D 2890          goto    AddrSendFail
008E 0064          clrwdt
008F 3401          retlw   TRUE
                                                   ;
                   _AddrSendFail:
0090 0064          clrwdt
0091 1F90          btfss   ACK_Error
0092 3400          retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                                                   ;
                                                   ; Address Not Acknowledged, so send STOP bit
                                                   ;
0093 205F          call    TxmtStopBit
0094 3400          retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                   ;
                   ;*****************************************************************************************************
                   ;             Transmit A Byte Of Data
                   ;
                   ; The data to be transmitted must be loaded into DataByte Reg
                   ; Clock stretching is allowed by slave. If the slave pulls the clock low,
                   ; then, the stretch is detected and INT Interrupt on Rising edge is enabled
                   ; and also RTCC timeout interrupt is enabled The clock stretching slows down
                   ; the transmit rate because all checking is done in
                   ; software. However, if the system has fast slaves and needs no clock
                   ; stretching, then this feature can be disabled during Assembly time by setting
```

© 1994 Microchip Technology Inc.

```
                        ;   _CLOCK_STRETCH_ENABLED must be set to FALSE.
                        ;
                        ;*****************************************************************************************************
                        SendData:
                        ;
                        ; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
                        ; For future compatibility, the user MUST call SendData & NOT TxmtByte
                        ;
0095 2896                  goto    TxmtByte
                                                        ;
                        TxmtByte:
0096 080E                  movf    DataByte,w
0097 0093                  movwf   DataByteCopy         ; make copy of DataByte
0098 1510                  bsf     Txmt_Progress        ; set Bus status for txmt progress
0099 1210                  bcf     Txmt_Success         ; reset status bit
009A 3008                  movlw   0x08
009B 008F                  movwf   BitCount
009C 1683                  bsf     rp0
                           if _CLOCK_STRETCH_CHECK
                                                        ; set RTCC to INT CLK timeout for 1 mSec
                                                        ; do not disturb user's selection of RPUB in OPTION Register
                                                        ;
009D 0801                  movf    option,w
009E 39C3                  andlw   OPTION_INIT          ; defined in I2C.H header file
009F 0081                  movwf   option
                           endif

                        TxmtNextBit:
00A0 0064                  clrwdt                       ; clear WDT, set for 18 mSec
00A1 1006                  bcf     SCL
00A2 0D93                  rlf     DataByteCopy         ; MSB first, Note DataByte Is Lost
00A3 1086                  bcf     SDA
00A4 1803                  btfsc   c
00A5 1486                  bsf     SDA
00A6 210B                  call    Delay47uSec          ; guarantee min LOW TIME tLOW & Setup time
00A7 1406                  bsf     SCL                  ; set clock high , check if clock is high, else
                                                        ; clock being stretched
00A8 210D                  call    Delay40uSec          ; guareentee min HIGH TIME tHIGH
                           if _CLOCK_STRETCH_CHECK
00A9 1283                  bcf     rp0
00AA 0181                  clrf    rtcc                 ; clear RTCC
00AB 110B                  bcf     rtif                 ; clear any pending flags
00AC 168B                  bsf     rtie                 ; elable RTCC Interrupt
00AD 1391                  bcf     TIME_OUT_            ; reset timeout error flag
                        Check_SCL_1:
00AE 1B91                  btfsc   TIME_OUT_            ; if RTCC timeout or Error then Abort & return
00AF 28FC                  goto    Bus_Fatal_Error      ; Possible FATAL Error on Bus
```


© 1994 Microchip Technology Inc.

```
00B0 1283          bcf     rp0
00B1 1C06          btfss   SCL                     ; if clock not being stretched, it must be high
00B2 28AE          goto    Check_SCL_1             ; loop until SCL high or RTCC timeout interrupt
00B3 128B          bcf     rtie                    ; Clock good, disable RTCC interrupts
00B4 1683          bsf     rp0

                   endif
00B5 0B8F          decfsz  BitCount
00B6 28A0          goto    TxmtNextBit
                                                   ;
                                                   ; Check For Acknowledge
                                                   ;
00B7 1006          bcf     SCL                     ; reset clock
00B8 1486          bsf     SDA                     ; Release SDA line for Slave to pull down
00B9 210B          call    Delay47uSec             ; guareentee min LOW TIME tLOW & Setup time
00BA 1406          bsf     SCL                     ; clock for slave to ACK
00BB 210D          call    Delay40uSec             ; guareentee min HIGH TIME tHIGH
00BC 1283          bcf     rp0                     ; select PAGE 0 to test PortB pin SDA
00BD 1886          btfsc   SDA                     ; SDA should be pulled low by slave if OK
00BE 28C5          goto    TxmtErrorAck
                                                   ;
00BF 1683          bsf     rp0
00C0 1006          bcf     SCL                     ; reset clock

00C1 1110          bcf     Txmt_Progress           ; reset TXMT bit in Bus Status
00C2 1610          bsf     Txmt_Success            ; transmission successful
00C3 1390          bcf     ACK_Error               ; ACK OK
00C4 0008          return
                   TxmtErrorAck:
                   RELEASE_BUS
00C5 1683
00C5 1486
00C6 1406
00C8 1110          bcf     Txmt_Progress           ; reset TXMT bit in Bus Status
00C9 1210          bcf     Txmt_Success            ; transmission NOT successful
00CA 1790          bsf     ACK_Error               ; No ACK From Slave
00CB 0008          return
                   ;
                   ;*****************************************************************************************************
                   ;
                   ;                Receive  A Byte Of Data From Slave
                   ;
                   ; assume address is already sent
                   ; if last byte to be received, do not acknowledge slave (last byte is
                   ; tested from Last_Byte_Rcv bit of control reg)
                   ; Data Received on successful reception is in DataReg register
                   ;
                   ;
```

© 1994 Microchip Technology Inc.

```
                ;*****************************************************************************************************
                ;
                GetData:
00CC 28CD           goto    RcvByte
                ;
                RcvByte:
00CD 1590           bsf     Rcv_Progress            ; set Bus status for txmt progress
00CE 1290           bcf     Rcv_Success             ; reset status bit
00CF 3008           movlw   0x08
00D0 008F           movwf   BitCount
                if _CLOCK_STRETCH_CHECK
00D1 1683           bsf     rp0
                                                    ; set RTCC to INT CLK timeout for 1 mSec
                                                    ; do not disturb user's selection of RPUB in OPTION Register
                                                    ;
00D2 0801           movf    option,w
00D3 39C3           andlw   OPTION_INIT             ; defined in I2C.H header file
00D4 0081           movwf   option
                      endif

                    RcvNextBit:
00D5 0064           clrwdt                          ; clear WDT, set for 18 mSec
00D6 1683           bsf     rp0                     ; page 1 for TRIS manipulation
00D7 1006           bcf     SCL
00D8 1486           bsf     SDA                     ; can be removed from loop
00D9 210B           call    Delay47uSec             ; guarantee min LOW TIME tLOW & Setup time
00DA 1406           bsf     SCL                     ; clock high, data sent by slave
00DB 210D           call    Delay40uSec             ; guarantee min HIGH TIME tHIGH
                        if _CLOCK_STRETCH_CHECK
00DC 1283           bcf     rp0
00DD 0181           clrf    rtcc                    ; clear RTCC
00DE 110B           bcf     rtif                    ; clear any pending flags
00DF 168B           bsf     rtie                    ; elable RTCC Interrupt
00E0 1391           bcf     TIME_OUT_               ; reset timeout error flag

                    Check_SCL_2:
00E1 1B91           btfsc   TIME_OUT_               ; if RTCC timeout or Error then Abort & return
00E2 28FC           goto    Bus_Fatal_Error         ; Possible FATAL Error on Bus
00E3 1283           bcf     rp0
00E4 1C06           btfss   SCL                     ; if clock not being stretched, it must be high
00E5 28E1           goto    Check_SCL_2             ; loop until SCL high or RTCC timeout interrupt
00E6 128B           bcf     rtie                    ; Clock good, diable RTCC interrupts
00E7 1683           bsf     rp0
                      endif
00E8 1283           bcf     rp0                     ; select page 0 to read Ports
00E9 1003           bcf     c
00EA 1886           btfsc   SDA
00EB 1403           bsf     c
```


© 1994 Microchip Technology Inc.

```
                ;                               ; TEMP ???? DO 2 out of 3 Majority detect
00EC 0D8E           rlf     DataByte            ; left shift data ( MSB first)
00ED 0B8F           decfsz  BitCount
00EE 28D5           goto    RcvNextBit
                                                ;
                                                ; Generate ACK bit if not last byte to be read,
                                                ; if last byte Generate NACK
                                                ; do not send ACK on last byte, main routine will send a STOP bit
00EF 1683           bsf     rp0
00F0 1006           bcf     SCL
00F1 1086           bcf     SDA                 ; ACK by pulling SDA low
00F2 1911           btfsc   Last_Byte_Rcv
00F3 1486           bsf     SDA                 ; if last byte, send NACK by setting SDA high
00F4 210B           call    Delay47uSec         ; guarantee min LOW TIME tLOW & Setup time
00F5 1406           bsf     SCL
00F6 210D           call    Delay40uSec         ; guarantee min HIGH TIME tHIGH
                      RcvEnd:
00F7 1006           bcf     SCL                 ; reset clock
00F8 1190           bcf     Rcv_Progress        ; reset TXMT bit in Bus Status
00F9 1690           bsf     Rcv_Success         ; transmission successful
00FA 1390           bcf     ACK_Error           ; ACK OK
00FB 0008       return

                if _CLOCK_STRETCH_CHECK
                ;*************************************************************************************************
                ;               Fatal Error On I2C Bus
                ;
                ;  Slave pulling clock for too long or if SCL Line is stuck low.
                ;  This occurs if during Transmission, SCL is stuck low for period longer
                ;  than approx. 1mS and RTCC times out (appox 4096 cycles : 256 * 16 -
                ;  prescaler of 16).
                    ;*********************************************************************************************
                    Bus_Fatal_Error:

                    ; disable RTCC Interrupt
                    ;
00FC 128B           bcf     rtie                    ; disable RTCC interrupts, until next TXMT try

                RELEASE_BUS
00FD 1683
00FD 1486
00FE 1406
                    ;
                    ;  Set the Bus_Status Bits appropriately
                    ;
0100 1490           bsf     Abort                   ; transmission was aborted
0101 1710           bsf     Fatal_Error             ; FATAL Error occured
0102 1110           bcf     Txmt_Progress           ; Transmission Is Not in Progress
0103 1210           bcf     Txmt_Success            ; Transmission Unsuccesful
```

© 1994 Microchip Technology Inc.

```
                        ;
0104 205F                   call    TxmtStopBit             ; Try sending a STOP bit, may be not successful
                        ;
0105 0008                   return
                        ;
                        ;*****************************************************************************************************
                            endif
                        ;*****************************************************************************************************
                        ;           General Purpose Delay Routines
                        ;
                        ;  Delay4uSis wait loop for 4.0 uSec
                        ;  Delay47uS   is wait loop for 4.7 uSec
                        ;  Delay50uS   is wait loop for 5.0 uSec
                        ;
                        ;*****************************************************************************************************
                        ;

                        Delay50uSec:
0106 3006                   movlw       ((_50uS_Delay-5)/3 + 1)
                            DlyK
0107 0092                   movwf       DelayCount
0108 0B92                   decfsz      DelayCount
0109 2908                   goto        $-1
010A 0008                   return
                        ;
                        Delay47uSec:
010B 3004                   movlw       ((47uS_Delay-8)/3 + 1)
010C 2907                   goto        DlyK
                        ;
                        Delay40uSec:
010D 3003                   movlw       ((_40uS_Delay-8)/3 + 1)
010E 2907                   goto        DlyK
                        ;
                        ;*****************************************************************************************************
                            endif
                        ;*****************************************************************************************************
                        ;

                        ReadSlave1:

                        ;
                        ;  EEPROM (24C04) may be in write mode (busy), check for ACK by sending a
                        ;  control byte
                        LOAD_ADDR_8  _Slave_1_Addr


010F 1011
010F 30A0
0110 008C
```


© 1994 Microchip Technology Inc.

```
                        wait1:
                    I2C_TEST_DEVICE

0112 2011
0113 1F11                   btfss   SlaveActive             ; See If slave is responding
0114 2912                   goto    wait1                   ; if stuck for ever, recover from WDT, can use other schemes
0115 0064                   clrwdt
                    I2C_READ_SUB  8, DataBegin+1, 0x50

0116 1091
0116 2057
0117 206B
0119 3050
0119 008E
011A 2095
011C 3007
011C 0096
011D 3021
011E 0084
0120 2039

                                                            ;
                                                            ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
                                                            ;
                    LOAD_ADDR_8   _Slave_2_Addr

0121 1011
0121 30AC
0122 008C

                        wait2:
                    I2C_TEST_DEVICE

0124 2011
0125 1F11                   btfss   SlaveActive             ; See If slave is responding
0126 2924                   goto    wait2                   ; if stuck for ever, recover from WDT, can use other schemes
0127 0064                   clrwdt

                    I2C_READ_SUB  8, DataBegin+1, 0x60

0128 1091
0128 2057
0129 206B
012B 3060
012B 008E
```


© 1994 Microchip Technology Inc.

```
012C 2095
012E 3007
012E 0096
012F 3021
0130 0084
0132 2039
0133 0008                       return
                                ;
                                ;*****************************************************************************************
                                ;
                                ReadSlave3:
                                ;
                                LOAD_ADDR_8  _Slave_3_Addr
                                ;
0134 1011
0134 30D6
0135 008C
                                ;
                                wait3:
                                I2C_TEST_DEVICE
                                ;
0137 2011
0138 1F11                       btfss   SlaveActive             ; See If slave is responding
0139 2937                       goto    wait3                   ; if stuck for ever, recover from WDT, can use other schemes
013A 0064                       clrwdt
                                I2C_READ_SUB  8, DataBegin, 0
                                ;
013B 1091
013B 2057
013C 206B
013E 3000
013E 008E
013F 2095
0141 3007
0141 0096
0142 3020
0143 0084
0145 2039
                                ;
                                ;
0146 0008                       return
                                ;
                                ;*****************************************************************************************
                                ;
                                ;       Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
                                ;
                                ;*****************************************************************************************
```


© 1994 Microchip Technology Inc.

```
                        FillDataBuf:

0147 3000               movlw   0x00                    ; start address location of EEPROM array
0148 00A0               movwf   DataBegin               ; 1st byte of data to be sent is start address
0149 3021               movlw   DataBegin+1             ; data starts following address (RAM Pointer)
014A 0084               movwf   fsr
014B 3008               movlw   8                       ; fill RAM with 8 bytes , this data is written to
                                                        ; EEPROM (slave)
014C 009A               movwf   byteCount
014D 3055               movlw   0x55                    ; pattern to fill with is 0x55 & 0xAA
014E 009B               movwf   HoldData
                        X1:
014F 099B               comf    HoldData
0150 081B               movf    HoldData,w
0151 0080               movwf   indf
0152 0A84               incf    fsr                     ; point to next location
0153 0B9A               decfsz  byteCount
0154 294F               goto    X1
0155 0008               return

                ;
                ;*****************************************************************************************************
                ;
                ;               Main Routine  (Test Program)
                ;
                ;       SINGLE  MASTER, MULTIPLE SLAVES
                ;
                ;*****************************************************************************************************

                        Start:
0156 204D               call    InitI2CBus_Master       ; initialize I2C Bus
0157 178B               bsf     gie                     ; enable global interrupts
                                                        ;

0158 2147               call    FillDataBuf             ; fill data buffer with 8 bytes of data (0x55, 0xAA)
                                                        ;
                ; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit
                ; Addr
                ; Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
                ;
0159 1810                       btfsc   Bus_Busy        ; is Bus Free, ie. has a start & stop  bit been
                                                        ; detected (only for multi master system)
015A 2959                       goto    $-1             ; a very simple test, unused for now

              LOAD_ADDR_8  _Slave_1_Addr

015B 1011
```

© 1994 Microchip Technology Inc.

```
015B 30A0
015C 008C

                I2C_WR      0x09, DataBegin

015E 3009
015E 0096
015F 3020
0160 0084
0162 2019
0162 205F

                                                ;
                                                ; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
                                                ;
0164 1810                 btfsc   Bus_Busy      ; is Bus Free, ie. has a start & stop bit been
                                                ; detected (only for multi master system)
0165 2964                 goto    $-1           ; a very simple test, unused for now

                LOAD_ADDR_8  _Slave_2_Addr

0166 1011
0166 30AC
0167 008C


                I2C_WR_SUB    0x08, DataBegin+1, 0x30

0169 3009
0169 0096
016B 3020
016B 0084
016D 0800
016D 0097
016E 3030
016F 0080
0171 2019
0172 0817
0172 00A0
0174 205F
0175 210F                 call    ReadSlave1    ; read a byte from slave from current address
                                                ;
                LOAD_ADDR_8  _Slave_3_Addr

0176 1011
0176 30D6
```


© 1994 Microchip Technology Inc.

```
0177 008C
0179 30CC                       movlw   0xCC
017A 00A0                       movwf   DataBegin
                        I2C_WR_SUB  0x01,DataBegin, 0x33

017B 3002
017B 0096
017D 301F
017D 0084
017F 0800
017F 0097
0180 3033
0181 0080
0183 2019
0184 0817
0184 009F
0186 205F
0187 2134                       call    ReadSlave3              ; Read From Slave PIC
                        ;

0188 0064                       self    clrwdt
0189 2988                       goto    self
                        ;
                        ;*****************************************************************************************************

                        END
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN578

Use of the SSP Module in the I²C™ Multi-Master Environment

INTRODUCTION

The Inter-IC (I²C) bus is a two-wire serial interface developed by Phillips/Signetics™. The specification supports data transmission up to 400 Kbps.

The I²C interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When the bus is active, one device is the Master (generates the clock and the handshaking signals), while all the other devices are Slaves. The current bus Master can both read from and write to any of the Slave units by addressing them individually. On a Multi-Master bus the Masters follow an arbitration scheme to ensure that the bus is not corrupted.

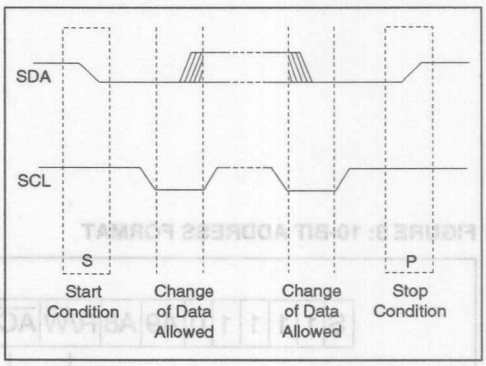
Each device attached to the I²C bus is assigned a unique address. When a Master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to "talk" to. All devices "listen" to see if this is their address. Within this address, a bit specifies whether the Master wishes to read from or write to the Slave device.

The output stages of each device on the bus attached to the clock (SCL) and data (SDA) lines must have an open-drain or open-collector in order to perform the wired-AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down. The only limitation on the number of devices that may be attached to the bus is limited by the maximum bus loading specification. For complete bus specifications, refer to Philips/Signetics™ document "The I²C-bus and how to use it". The order number for this document is 98-8080-575.

INITIATING AND TERMINATING DATA TRANSFER

During times of no data transfer (idle time), both SCL and SDA are pulled high. A Master device which wishes to take control of the bus must first generate a START condition. The START condition is defined as a high to low transition of SDA when SCL is high. When the Master has completed all data transmissions and wishes to relinquish the bus, it generates a STOP condition. The STOP condition is defined as a low to high transition of SDA when SCL is high. Because the START and STOP conditions are defined as transitions of the SDA when the SCL is high, the SDA line can only change when SCL is low during the actual data transmission. Figure 1 shows the relationship between SCL and SDA for the various conditions.

FIGURE 1: START AND STOP CONDITIONS



3

© 1994 Microchip Technology Inc.

ADDRESSING I²C DEVICES

There are two address formats. The simplest is the 7-bit address format with a R/W bit (see Figure 2). The more complex is the 10-bit address with a R/W bit (see Figure 3). For 10-bit addressing, two bytes must be transmitted with the first five bits specifying this to be a 10-bit address. Only 7-bit addressing is used in this application note.

FIGURE 2: 7-BIT ADDRESS FORMAT

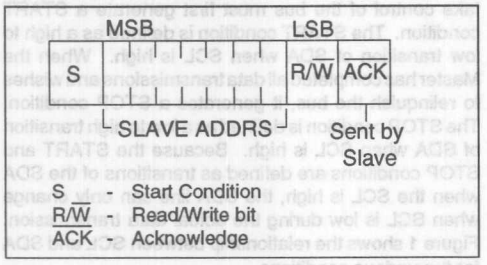
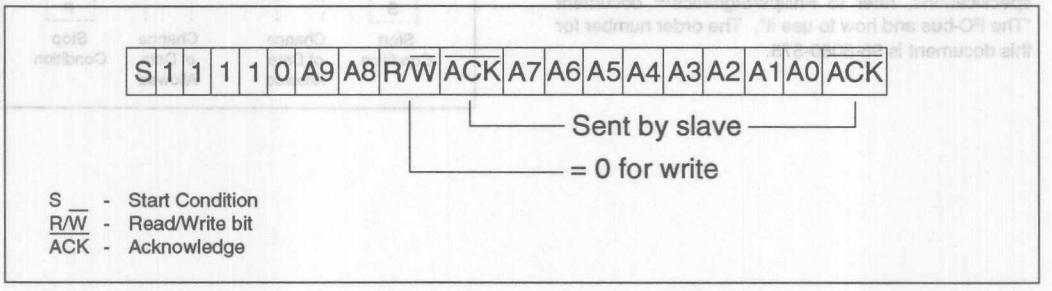


FIGURE 3: 10-BIT ADDRESS FORMAT



TRANSFER ACKNOWLEDGE

Slave as receiver:

All data is transmitted as bytes, with no limit to the number of bytes transmitted per data transfer. After each byte, the slave-receiver generates an acknowledge bit (ACK) by pulling the SDA line low. When a slave-receiver doesn't acknowledge the slave address or received data, the master aborts the transfer. Whether the ACK bit is generated or not, SDA must be released by the slave so that the master can generate the STOP condition.

Master as receiver:

If the master is receiving the data, it generates an acknowledge signal for each received byte of data except for the last byte. To signal the end of data to the slave-transmitter, the master does not generate an acknowledge. The slave then releases the SDA line so the master can generate the STOP condition. The master can also generate the STOP condition during the acknowledge pulse for valid termination of data transfer.

If the slave needs to delay the transmission of the next byte, holding the SCL line low will force the master into a wait state. Data transfer continues when the slave releases the SCL line. This allows the slave to move the received data or fetch the data it needs to transfer before allowing the clock to start. This wait state technique can also be implemented at the bit level.

© 1994 Microchip Technology Inc.

MULTI-MASTER

The I²C protocol allows a system to have more than one master. When two or more masters try to transfer data at the same time, arbitration and synchronization occur.

Arbitration:

Arbitration takes place on the SDA line while the SCL line is high. The master which transmits a high when the other master transmits a low loses arbitration (see Figure 4) and turns off its data output stage. A master which lost arbitration can generate clock pulses until the end of the data byte where it lost arbitration. When the master devices are addressing the same device, arbitration continues into the data.

Clock Synchronization:

Clock synchronization occurs after the devices have started arbitration. This is performed using a wired-AND connection to the SCL line. A high to low transition on the SCL line causes the concerned devices to start counting off their low period. Once a device clock has gone low, it will hold the SCL line low until its SCL high state is reached. The low to high transition of this clock may not change the state of the SCL line, if another device clock is still within its low period. The SCL line is held low by the device with the longest low period. Devices with shorter low periods enter a high wait-state, until the SCL line comes high. When the SCL line comes high, all devices start counting off their high periods. The first device to complete its high period will pull the SCL line low. The SCL line high time is determined by the device with the shortest high period. This is shown in Figure 5.

3

FIGURE4: MULTI-MASTER ARBITRATION

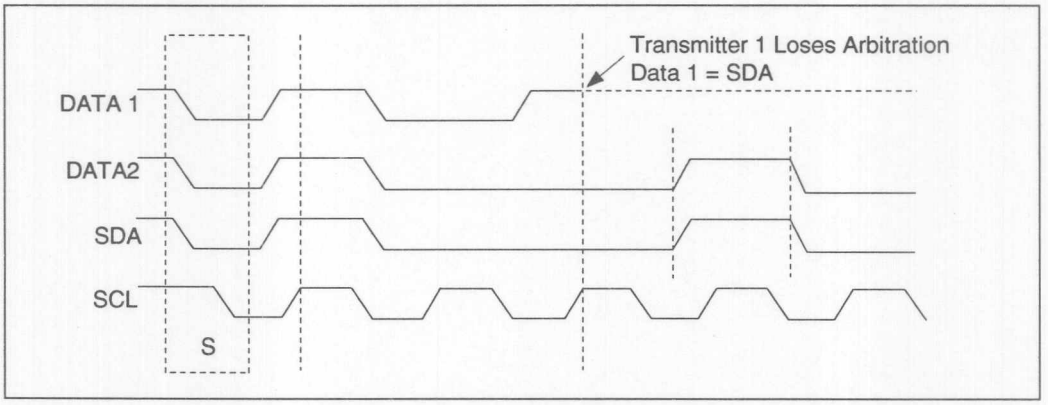
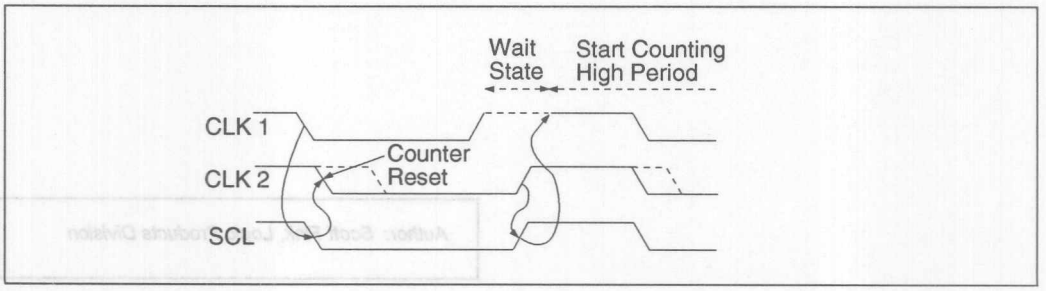


FIGURE 5: CLOCK SYNCHRONIZATION



© 1994 Microchip Technology Inc.

IMPLEMENTATION IN THE PIC16CXX

This Application Note uses the PIC16CXX in a Multi-Master I²C environment. The PIC16CXX acts as both a Master and a Slave on the bus.

Hardware:

The demonstration hardware consists of a keypad multiplexed with eight LEDs on Port B and connections to the I²C through the RC3/SCL and RC4/SDA pins. (See Figure 6).

Software:

The software transmits in Master mode and receives data in Slave mode.

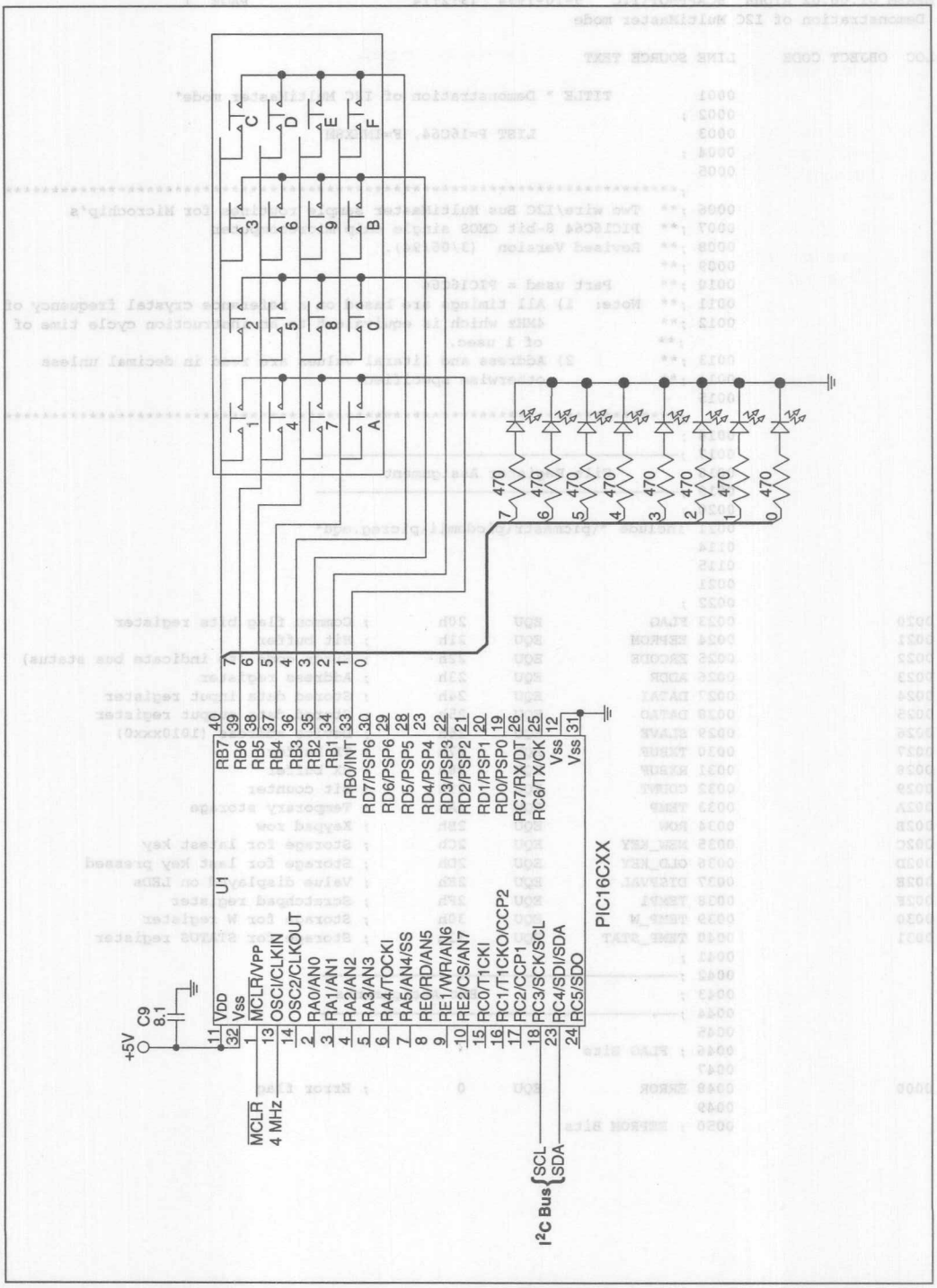
The transmit routine scans the keypad, debounces the keypresses, and transmits their encoded value in Master mode over the I²C bus to slave address A6. Before transmitting, the status of the Synchronous Serial Port (SSP) is checked . No data is sent until the SSP status register indicates that a Stop bit was received. If the transmission causes any errors, the error code will be displayed on the LEDs, and transmission will again be attempted.

Data received in Slave mode at address A2 from the bus is displayed on the LEDs. Slave reception is interrupt driven. When a complete word is received with the proper address, the processor is interrupted, and the program verifies that an SSP interrupt was received. Once it has been verified that the interrupt was caused by the SSP, and that the buffer is full, the data word is read and output to the LEDs.

Author: Scott Fink, Logic Products Division

© 1994 Microchip Technology Inc.

FIGURE 6: HARDWARE USED FOR THIS AP-NOTE



```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14                    PAGE  1
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         TITLE " Demonstration of I2C MultiMaster mode"
                     0002 ;
                     0003                 LIST P=16C64, F=INHX8M
                     0004 ;
                     0005
                         ;*****************************************************************************
                     0006 ;**  Two wire/I2C Bus MultiMaster sample routines for Microchip's
                     0007 ;**  PIC16C64 8-bit CMOS single chip microcomputer
                     0008 ;**  Revised Version  (3/06/94).
                     0009 ;**
                     0010 ;**      Part used = PIC16C64
                     0011 ;**  Note:  1) All timings are based on a reference crystal frequency of
                     0012 ;**             4MHz which is equivalent to an instruction cycle time of
                          ;**             of 1 usec.
                     0013 ;**          2) Address and literal values are read in decimal unless
                     0014 ;**             otherwise specified.
                     0015
                         ;*****************************************************************************
                     0016 ;
                     0017 ;-----------------------------------------
                     0018 ;       File Register Assignment
                     0019 ;-----------------------------------------
                     0020 ;
                     0021 include "\picmastr\picdemii\picreg.equ"
                     0114
                     0115
                     0021
                     0022 ;
0020                 0023 FLAG            EQU     20h        ; Common flag bits register
0021                 0024 EEPROM          EQU     21h        ; Bit buffer
0022                 0025 ERCODE          EQU     22h        ; Error code (to indicate bus status)
0023                 0026 ADDR            EQU     23h        ; Address register
0024                 0027 DATAI           EQU     24h        ; Stored data input register
0025                 0028 DATAO           EQU     25h        ; Stored data output register
0026                 0029 SLAVE           EQU     26h        ; Device address (1010xxx0)
0027                 0030 TXBUF           EQU     27h        ; TX buffer
0028                 0031 RXBUF           EQU     28h        ; RX buffer
0029                 0032 COUNT           EQU     29h        ; Bit counter
002A                 0033 TEMP            EQU     2Ah        ; Temporary storage
002B                 0034 ROW             EQU     2Bh        ; Keypad row
002C                 0035 NEW_KEY         EQU     2Ch        ; Storage for latest key
002D                 0036 OLD_KEY         EQU     2Dh        ; Storage for last key pressed
002E                 0037 DISPVAL         EQU     2Eh        ; Value displayed on LEDs
002F                 0038 TEMP1           EQU     2Fh        ; Scratchpad register
0030                 0039 TEMP_W          EQU     30h        ; Storage for W register
0031                 0040 TEMP_STAT       EQU     31h        ; Storage for STATUS register
                     0041 ;
                     0042 ;-----------------------------------------
                     0043 ;                  Bit Assignments
                     0044 ;-----------------------------------------
                     0045
                     0046 ; FLAG Bits
                     0047
0000                 0048 ERROR           EQU     0          ; Error flag
                     0049
                     0050 ; EEPROM Bits
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  2
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

                       0051
0007                   0052 DI         EQU     7               ; EEPROM input
0006                   0053 DO         EQU     6               ; EEPROM output
                       0054
                       0055 ; I2C Device Bits
                       0056
0004                   0057 SDA        EQU     4               ; RB7, data in/out
0003                   0058 SCL        EQU     3               ; RB6, serial clock
                       0059
                       0060 ;End of files/bits equate
                       0061
                       0062            ORG     00h             ; Reset Vector
0000 2810              0063            goto    starting
                       0064
                       0065            ORG     04h             ; Interrupt Vector
0004 28EB              0066            goto    service_int
                       0067
                       0068            ORG     10h             ; Begining of Program space
                       0069 starting
0010 01AE              0070            clrf    DISPVAL         ; Blank out LEDs
0011 01A0              0071            clrf    FLAG            ; Clear error register
0012 0187              0072            clrf    PORT_C          ; Set SDA, SCL low when not is tri-state
0013 3008              0073            movlw   08h
0014 00AB              0074            movwf   ROW             ; Set initial row to be strobed
0015 303E              0075            movlw   B'00111110'     ; I2C 7 bit slave mode with master
0016 0094              0076            movwf   SSPCON          ;  mode enabled
0017 1683              0077            bsf     STATUS,RP0      ; Select page 1
0018 0186              0078            clrf    TRIS_B          ; Set PORT_B to all outputs
0019 3008              0079            movlw   B'00001000'     ; Enable SSP interrupt
001A 008C              0080            movwf   PIE1
001B 30A2              0081            movlw   b'10100010'     ; Slave address
001C 0093              0082            movwf   SSPADD
001D 1587              0083            bsf     TRIS_C,3        ; Set SCL high
001E 1607              0084            bsf     TRIS_C,4        ; Set SDA high
001F 1283              0085            bcf     STATUS,RP0      ; Select page 0
0020 118C              0086            bcf     PIR1,3          ; Clear SSP interrupt flag
0021 30C0              0087            movlw   B'11000000'     ; Enable interrupts
0022 008B              0088            movwf   INTCON
                       0089
                       0090 KbdWait
0023 01AA              0091            clrf    TEMP
0024 216B              0092            call    SetupDelay
0025 2102              0093            call    ScanKbd         ; Check for key pressed
0026 1283              0094            bcf     STATUS,RP0
0027 082C              0095            movf    NEW_KEY,W       ; Get latest key
0028 39FF              0096            andlw   0FFh            ; Key pressed?
0029 1903              0097            btfsc   STATUS,Z
002A 2823              0098            goto    KbdWait         ; No, go check again
002B 00A5              0099            movwf   DATAO           ; Yes, output it on I2C bus
002C 30A6              0100            movlw   B'10100110'     ; Address of device being addressed
002D 00A6              0101            movwf   SLAVE
002E 1683              0102            bsf     STATUS,RP0
                       0103 CheckAgain
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  3
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

002F 1A14            0104          btfsc   SSPSTAT,4       ;If STOP bit received last...
0030 2833            0105          goto    Goxmit          ;  OK to transmit
0031 1994            0106          btfsc   SSPSTAT,3       ;If START bit recieved last...
0032 282F            0107          goto    CheckAgain      ;  wait for STOP bit
                     0108 Goxmit
0033 1283            0109          bcf     STATUS,RP0
0034 138B            0110          bcf     INTCON,7        ; Disable interrupts
0035 203F            0111          call    WRBYTE          ; Output byte
0036 1C20            0112          btfss   FLAG,ERROR      ; Check for error
0037 283D            0113          goto    Checkout        ; No error, go on
0038 0822            0114          movf    ERCODE,W        ; Get error code
0039 0086            0115          movwf   PORT_B          ; Put error code on LEDs
003A 00AE            0116          movwf   DISPVAL
003B 1020            0117          bcf     FLAG,ERROR      ; Clear error flag
003C 282F            0118          goto    CheckAgain
                     0119 Checkout
003D 178B            0120          bsf     INTCON,7        ; Enable interrupts
003E 2823            0121          goto    KbdWait
                     0122
                     0123 ;*****************************************
                     0124 ;       BYTE-WRITE, write one byte to I2C (Master Mode)
                     0125 ;*****************************************
                     0126 ;       Input   :       DATAO= data to be written
                     0127 ;                       ADDR    = destination address
                     0128 ;                       SLAVE   = device address (1010xxx0)
                     0129 ;       Output  :       Data written to EEPROM device
                     0130 ;*****************************************
                     0131 ;
                     0132 WRBYTE
003F 1283            0133          bcf     STATUS,RP0
0040 0826            0134          movf    SLAVE,W         ; Send SLAVE address
0041 00A7            0135          movwf   TXBUF           ;  to TX buffer
0042 20BF            0136          call    BSTART          ; Generate START bit
0043 2071            0137          call    TX              ; Output SLAVE data address
0044 1283            0138          bcf     STATUS,RP0
0045 0825            0139          movf    DATAO,W         ; Move DATA
0046 00A7            0140          movwf   TXBUF           ;  into transmit buffer
0047 2071            0141          call    TX              ; Output DATA and detect
                                                           ; acknowledgement
0048 20CD            0142          call    BSTOP           ; Generate STOP bit
0049 0008            0143          return
                     0144
                     0145 ;*****************************************
                     0146 ;       BYTE-READ, read one byte from I2C (Master Mode)
                     0147 ;*****************************************
                     0148 ;       Input   :       ADDR    = source address
                     0149 ;                       SLAVE   = device address (1010xxx0)
                     0150 ;       Output  :       DATAI   = data read from serial EEPROM
                     0151 ;*****************************************
                     0152
                     0153 RDBYTE
004A 1283            0154          bcf     STATUS,RP0
004B 0826            0155          movf    SLAVE,W         ; Move SLAVE address
004C 00A7            0156          movwf   TXBUF           ;  into buffer (R/W = 0)
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14                PAGE  4
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

004D 20BF            0157         call    BSTART          ; Generate START bit
004E 2071            0158         call    TX              ; Output SLAVE address. Check ACK.
004F 1283            0159         bcf     STATUS,RP0
0050 0823            0160         movf    ADDR,W          ; Put slave data address into
0051 00A7            0161         movwf   TXBUF           ;  Xmit buffer
0052 2071            0162         call    TX              ; Output WORD address. Check ACK.
0053 20BF            0163         call    BSTART          ; START READ
0054 1283            0164         bcf     STATUS,RP0
0055 0826            0165         movf    SLAVE,W         ;
0056 00A7            0166         movwf   TXBUF
0057 1427            0167         bsf     TXBUF,0         ; Specify READ mode (R/W = 1)
0058 2071            0168         call    TX              ; Output SLAVE address
0059 205F            0169         call    RX              ; READ in data and acknowledge
005A 20CD            0170         call    BSTOP           ; Generate STOP bit
005B 1283            0171         bcf     STATUS,RP0
005C 0828            0172         movf    RXBUF,W         ; Save data from buffer
005D 00A4            0173         movwf   DATAI           ; to DATAI file.
005E 0008            0174         return
                     0175
                     0176 ;-----------------------------------------------
                     0177 ;       RECEIVE eight data bits subroutine
                     0178 ;-----------------------------------------------
                     0179 ;       Input   :       None
                     0180 ;       Output  :       RXBUF = 8-bit data received
                     0181 ;-----------------------------------------------
                     0182
                     0183 RX
005F 1283            0184         bcf     STATUS,RP0
0060 3008            0185         movlw   .8              ; 8 bits of data
0061 00A9            0186         movwf   COUNT
0062 01A8            0187         clrf    RXBUF
                     0188 ;
                     0189 RXLP
0063 0DA8            0190         rlf     RXBUF           ; Shift data to buffer
0064 1C03            0191         btfss   3,0
0065 1028            0192         bcf     RXBUF,0         ; carry -> f(0)
0066 1803            0193         btfsc   3,0
0067 1428            0194         bsf     RXBUF,0
0068 2087            0195         call    BITIN
0069 1283            0196         bcf     STATUS,RP0
006A 1BA1            0197         btfsc   EEPROM,DI
006B 1428            0198         bsf     RXBUF,0         ; Input bit =1
006C 0BA9            0199         decfsz  COUNT           ; 8 bits?
006D 2863            0200         goto    RXLP
006E 1721            0201         bsf     EEPROM,DO       ; Set acknowledge bit = 1
006F 209E            0202         call    BITOUT          ; to STOP further input
0070 3400            0203         retlw   0
                     0204
                     0205 ;-----------------------------------------------
                     0206 ;       TRANSMIT 8 data bits subroutine
                     0207 ;-----------------------------------------------
                     0208 ;       Input   :       TXBUF
                     0209 ;       Output  :       Data X'mitted to EEPROM device
```

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  5
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT
                     0210 ;-------------------------------------
                     0211
                     0212 TX
0071 1283            0213          bcf     STATUS,RP0
0072 3008            0214          movlw   .8
0073 00A9            0215          movwf   COUNT
                     0216 ;
                     0217 TXLP
0074 1321            0218          bcf     EEPROM,DO       ; Shift data bit out.
0075 1BA7            0219          btfsc   TXBUF,7         ; If shifted bit = 0, data bit = 0
0076 1721            0220          bsf     EEPROM,DO       ; Otherwise data bit = 1
0077 209E            0221          call    BITOUT          ; Serial data out
0078 1283            0222          bcf     STATUS,RP0
0079 0DA7            0223          rlf     TXBUF           ; Rotate TXBUF left
007A 1C03            0224          btfss   3,0             ; f(6) -> f(7)
007B 1027            0225          bcf     TXBUF,0         ; f(7) -> carry
007C 1803            0226          btfsc   3,0             ; carry -> f(0)
007D 1427            0227          bsf     TXBUF,0
007E 0BA9            0228          decfsz  COUNT           ; 8 bits done?
007F 2874            0229          goto    TXLP            ; No.
0080 2087            0230          call    BITIN           ; Read acknowledge bit
0081 1283            0231          bcf     STATUS,RP0
0082 3003            0232          movlw   3
0083 1BA1            0233          btfsc   EEPROM,DI       ; Check for acknowledgement
0084 20DE            0234          call    ERR             ; No acknowledge from device
0085 1283            0235          bcf     STATUS,RP0
0086 3400            0236          retlw   0
                     0237
                     0238 ;-------------------------------------
                     0239 ;       Single bit receive from I2C to PIC
                     0240 ;-------------------------------------
                     0241 ;       Input   :       None
                     0242 ;       Output  :       Data bit received
                     0243 ;-------------------------------------
                     0244
                     0245 BITIN
0087 1683            0246          bsf     STATUS,RP0
0088 1607            0247          bsf     TRIS_C,SDA      ; Set SDA for input
0089 1283            0248          bcf     STATUS,RP0
008A 13A1            0249          bcf     EEPROM,DI
008B 1683            0250          bsf     STATUS,RP0
008C 1587            0251          bsf     TRIS_C,SCL      ; Clock high
008D 3001            0252          movlw   1
008E 1283            0253          bcf     STATUS,RP0
008F 1987            0254          btfsc   PORT_C,SCL      ; Skip if SCL  is high
0090 2895            0255          goto    BIT1
0091 1283            0256          bcf     STATUS,RP0
0092 1C20            0257          btfss   FLAG,ERROR      ; Remain as first error encountered
0093 00A2            0258          movwf   ERCODE          ; Save error code
0094 1420            0259          bsf     FLAG,ERROR      ; Set error flag
                     0260 BIT1
0095 1283            0261          bcf     STATUS,RP0
0096 1E07            0262          btfss   PORT_C,SDA      ; Read SDA pin, for ACK low
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  6
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

0097 289A            0263          goto    ACKOK
0098 1283            0264          bcf     STATUS,RP0
0099 17A1            0265          bsf     EEPROM,DI         ; DI = 1
                     0266 ACKOK
009A 1683            0267          bsf     STATUS,RP0
009B 0000            0268          nop                       ; Delay
009C 1187            0269          bcf     TRIS_C,SCL        ; Return SCL to low
009D 3400            0270          retlw   0
                     0271
                     0272 ;-------------------------------------------
                     0273 ;       Single bit data transmit from PIC to I2C
                     0274 ;-------------------------------------------
                     0275 ;       Input   :       EEPROM register, bit DO
                     0276 ;       Output  :       Bit transmitted over I2C
                     0277 ;                       Error bits set as necessary
                     0278 ;-------------------------------------------
                     0279
                     0280 BITOUT
009E 1283            0281          bcf     STATUS,RP0
009F 1F21            0282          btfss   EEPROM,DO
00A0 28AB            0283          goto    BIT0
00A1 1683            0284          bsf     STATUS,RP0
00A2 1607            0285          bsf     TRIS_C,SDA        ; Output bit 0
00A3 3002            0286          movlw   2
00A4 1283            0287          bcf     STATUS,RP0
00A5 1A07            0288          btfsc   PORT_C,SDA        ; Check for error code 2
00A6 28B0            0289          goto    CLK1
00A7 1C20            0290          btfss   FLAG,ERROR        ; Remain as first error encountered
00A8 00A2            0291          movwf   ERCODE            ; Save error code
00A9 1420            0292          bsf     FLAG,ERROR        ; Set error flag
00AA 28B0            0293          goto    CLK1              ; SDA locked low by device
                     0294 ;
                     0295 BIT0
00AB 1683            0296          bsf     STATUS,RP0
00AC 1207            0297          bcf     TRIS_C,SDA        ; Output bit 0
00AD 0000            0298          nop                       ; Delay
00AE 0000            0299          nop
00AF 0000            0300          nop
                     0301 CLK1
00B0 1683            0302          bsf     STATUS,RP0
00B1 1587            0303          bsf     TRIS_C,SCL
00B2 3001            0304          movlw   1                 ; Error code 1
00B3 1283            0305          bcf     STATUS,RP0
00B4 1987            0306          btfsc   PORT_C,SCL        ; SCL locked low?
00B5 28BA            0307          goto    BIT2              ; No.
00B6 1283            0308          bcf     STATUS,RP0
00B7 1C20            0309          btfss   FLAG,ERROR        ; Yes.
00B8 00A2            0310          movwf   ERCODE            ; Save error code
00B9 1420            0311          bsf     FLAG,ERROR        ; Set error flag
                     0312 BIT2
00BA 0000            0313          nop
00BB 0000            0314          nop
00BC 1683            0315          bsf     STATUS,RP0
```

3

© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  7
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

00BD 1187            0316         bcf     TRIS_C,SCL      ; Return SCL to low
00BE 3400            0317         retlw   0
                     0318
                     0319 ;-------------------------------------------
                     0320 ;       START bit generation routine
                     0321 ;-------------------------------------------
                     0322 ;       input   : none
                     0323 ;       output  : initialize bus communication
                     0324 ;-------------------------------------------
                     0325
                     0326 ;Generate START bit (SCL is high while SDA goes from high to low
                   ; transition)
                     0327 ;and check status of the serial clock.
                     0328 BSTART
00BF 1683            0329         bsf     STATUS,RP0
00C0 1607            0330         bsf     TRIS_C,SDA      ; Make sure SDA is high
00C1 1587            0331         bsf     TRIS_C,SCL      ; Set clock high
00C2 3001            0332         movlw   1               ; Ready error status code 1
00C3 1283            0333         bcf     STATUS,RP0
00C4 1D87            0334         btfss   PORT_C,SCL      ; Locked?
00C5 20DE            0335         call    ERR             ; SCL locked low by device, flag error
00C6 1683            0336         bsf     STATUS,RP0
00C7 1207            0337         bcf     TRIS_C,SDA      ; SDA goes low during SCL high
00C8 0000            0338         nop                     ; Timing adjustment, 1.5uS @2MHz
00C9 0000            0339         nop
00CA 0000            0340         nop
00CB 1187            0341         bcf     TRIS_C,SCL      ; Start clock train
00CC 3400            0342         retlw   0
                     0343
                     0344 ;-------------------------------------------
                     0345 ;       STOP bit generation routine
                     0346 ;-------------------------------------------
                     0347 ;       Input   :       None
                     0348 ;       Output  :       Bus communication, STOP condition
                     0349 ;-------------------------------------------
                     0350
                     0351 ;Generate STOP bit (SDA goes from low to high during SCL high state)
                     0352 ;and check bus conditions.
                     0353
                     0354 BSTOP
00CD 1683            0355         bsf     STATUS,RP0
00CE 1207            0356         bcf     TRIS_C,SDA      ; Return SDA to low
00CF 1587            0357         bsf     TRIS_C,SCL      ; Set SCL high
00D0 0000            0358         nop
00D1 0000            0359         nop
00D2 0000            0360         nop
00D3 3001            0361         movlw   1               ; Ready error code 1
00D4 1283            0362         bcf     STATUS,RP0
00D5 1D87            0363         btfss   PORT_C,SCL      ;High?
00D6 20DE            0364         call    ERR             ; No, SCL locked low by device
00D7 1683            0365         bsf     STATUS,RP0
00D8 1607            0366         bsf     TRIS_C,SDA      ; SDA goes from low to high during SCL high
00D9 3004            0367         movlw   4               ; Ready error code 4
00DA 1E07            0368         btfss   TRIS_C,SDA      ;High?
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  8
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

00DB 20DE            0369         call    ERR             ; No, SDA bus not release for STOP
00DC 1283            0370         bcf     STATUS,RP0
00DD 3400            0371         retlw   0
                     0372
                     0373 ;-------------------------------------------------
                     0374 ; Two wire/I2C - CPU communication error status table and subroutine
                     0375 ;-------------------------------------------------
                     0376 ;  input  :    W-reg   = error code
                     0377 ;  output :    ERCODE  = error code
                     0378 ;           FLAG(ERROR) = 1
                     0379 ;
                     0380 ;         code          error status mode
                     0381 ;         ----          -----------------
                     0382 ;           1   :   SCL locked low by device (bus is still busy)
                     0383 ;           2   :   SDA locked low by device (bus is still busy)
                     0384 ;           3   :   No acknowledge from device (no handshake)
                     0385 ;           4   :   SDA bus not released for master to generate STOP bit
                     0386 ;-------------------------------------------------
                     0387 ;
                     0388 ;Subroutine to identify the status of the serial clock (SCL) and serial
                     0389 ;data (SDA) condition according to the error status table. Codes
                     0390 ;generated are useful for bus/device diagnosis.
                     0391
                     0392 ERR
00DE 1283            0393         bcf     STATUS,RP0
00DF 1C20            0394         btfss   FLAG,ERROR      ; Keep first error reported
00E0 00A2            0395         movwf   ERCODE          ; Save error code
00E1 1420            0396         bsf     FLAG,ERROR      ; Set error flag
00E2 3400            0397         retlw   0
                     0398
                     0399 ;-------------------------------------------------
                     0400 ;       DELAY, Provide a 1.54mS delay (@ 4 MHz clock)
                     0401 ;-------------------------------------------------
                     0402 ;       Input   :       None
                     0403 ;       Output  :       None
                     0404 ;-------------------------------------------------
                     0405
                     0406 delay
00E3 1283            0407         bcf     STATUS,RP0
00E4 01AA            0408         clrf    TEMP            ;clear last location
                     0409 dly1
00E5 0000            0410         nop
00E6 0000            0411         nop
00E7 0000            0412         nop
00E8 0BAA            0413         decfsz  TEMP            ;reduce count
00E9 28E5            0414         goto    dly1            ;loop
00EA 3400            0415         retlw   0
                     0416
                     0417 ;-------------------------------------------------
                     0418 ; Interrupt service routine
                     0419 ;  Only the SSP interrupt is enabled.  This routine will read the I2C
                     0420 ;  data and output it on the LEDs.
                     0421 ;-------------------------------------------------
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE  9
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0422
                     0423 service_int
00EB 118C            0424         bcf     PIR1,3          ; Clear SSP interrupt
00EC 00B0            0425         movwf   TEMP_W          ; Save W register
00ED 0E03            0426         swapf   STATUS,W        ; Get STATUS register
00EE 00B1            0427         movwf   TEMP_STAT       ; Save STATUS register
00EF 1683            0428         bsf     STATUS,RP0
00F0 1C14            0429         btfss   SSPSTAT,0       ; Check Buffer Full Flag
00F1 28FA            0430         goto    IntOut          ; No data received, so exit
00F2 1283            0431         bcf     STATUS,RP0
00F3 0813            0432         movf    SSPBUF,W        ; Get I2C data
00F4 1683            0433         bsf     STATUS,RP0
00F5 1E94            0434         btfss   SSPSTAT,5       ; If Address received last...
00F6 28FA            0435         goto    IntOut          ;  exit without saving it
00F7 1283            0436         bcf     STATUS,RP0
00F8 0086            0437         movwf   PORT_B          ; Display received data on LEDs
00F9 00AE            0438         movwf   DISPVAL
                     0439 IntOut
00FA 1683            0440         bsf     STATUS,RP0
00FB 158C            0441         bsf     PIE1,3          ; Re-enable SSP interrupt
00FC 1283            0442         bcf     STATUS,RP0
00FD 0E31            0443         swapf   TEMP_STAT,W     ; Restore STATUS register
00FE 0083            0444         movwf   STATUS
00FF 0EB0            0445         swapf   TEMP_W,1
0100 0E30            0446         swapf   TEMP_W,W        ; Restore W register
0101 0009            0447         retfie
                     0448
                     0449 ;-------------------------------------
                     0450 ; Keyboard scan routine
                     0451 ;  This routine scans the keypad connected to PORT_B, and
                     0452 ;  returns the pressed key in New_Key.
                     0453 ;-------------------------------------
                     0454
                     0455 ScanKbd
0102 1283            0456         bcf     STATUS,RP0
0103 01AC            0457         clrf    NEW_KEY         ; Clear key register
0104 082D            0458         movf    OLD_KEY,w       ; If key was pressed last pass through
0105 1D03            0459         btfss   STATUS,Z        ;  goto Debounce
0106 291C            0460         goto    Debounce
                     0461 Kbdloop
0107 1003            0462         bcf     STATUS,C
0108 0DAB            0463         rlf     ROW             ; Select next row to strobe
0109 1C03            0464         btfss   STATUS,C
010A 290D            0465         goto    Notdone
010B 3010            0466         movlw   010h            ; Start over at first row
010C 00AB            0467         movwf   ROW
                     0468 Notdone
010D 0186            0469         clrf    PORT_B
010E 1683            0470         bsf     STATUS,RP0
010F 300F            0471         movlw   00Fh            ; Set PORT_B for keypad read
0110 0086            0472         movwf   TRIS_B
0111 1283            0473         bcf     STATUS,RP0
0112 082B            0474         movf    ROW,W           ; Output Row
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14                PAGE 10
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

0113 0086            0475          movwf   PORT_B          ; Read colums
0114 300F            0476          movlw   0Fh             ; Mask out rows
0115 0506            0477          andwf   PORT_B,W        ; Check for Key press
0116 1903            0478          btfsc   STATUS,Z
0117 2933            0479          goto    KBDOUT          ; No key pressed, exit
0118 0186            0480          clrf    PORT_B
0119 042B            0481          iorwf   ROW,W           ; Key pressed, save it
011A 00AC            0482          movwf   NEW_KEY
011B 00AD            0483          movwf   OLD_KEY
                     0484 Debounce
011C 0186            0485          clrf    PORT_B
011D 1683            0486          bsf     STATUS,RP0
011E 0186            0487          clrf    TRIS_B
011F 1283            0488          bcf     STATUS,RP0
0120 082E            0489          movf    DISPVAL,W       ; Set LEDs
0121 0086            0490          movwf   PORT_B
0122 01AA            0491          clrf    TEMP
0123 216B            0492          call    SetupDelay
0124 216B            0493          call    SetupDelay      ; Delay for key debounce
0125 0186            0494          clrf    PORT_B
0126 1683            0495          bsf     STATUS,RP0
0127 300F            0496          movlw   0Fh             ; Set PORT_B for keypad read
0128 0086            0497          movwf   TRIS_B
0129 1283            0498          bcf     STATUS,RP0
012A 082B            0499          movf    ROW,W
012B 0086            0500          movwf   PORT_B          ; Output Row
012C 082D            0501          movf    OLD_KEY,W
012D 0606            0502          xorwf   PORT_B,w        ; Compare key with last key pressed
012E 1903            0503          btfsc   STATUS,Z
012F 2933            0504          goto    KBDOUT
0130 0186            0505          clrf    PORT_B          ; Key released, clear registers
0131 01AC            0506          clrf    NEW_KEY
0132 01AD            0507          clrf    OLD_KEY
                     0508 KBDOUT
0133 082E            0509          movf    DISPVAL,W       ; Set LEDs
0134 0086            0510          movwf   PORT_B
0135 1683            0511          bsf     STATUS,RP0
0136 0186            0512          clrf    TRIS_B
0137 1283            0513          bcf     STATUS,RP0
                     0514 KEY_DEC
0138 300F            0515          movlw   00Fh
0139 052C            0516          andwf   NEW_KEY,W       ; Get column of key pressed
013A 00AA            0517          movwf   TEMP
013B 1903            0518          btfsc   STATUS,Z        ; If no key pressed, exit
013C 2955            0519          goto    DECOUT
013D 30FF            0520          movlw   0FFh            ; Initialize the W register
                     0521 DECL1
013E 3E01            0522          addlw   001h            ; Count column
013F 0CAA            0523          rrf     TEMP            ; Rotate column until it is found
0140 1C03            0524          btfss   STATUS,C
0141 293E            0525          goto    DECL1
0142 00AA            0526          movwf   TEMP
0143 0E2C            0527          swapf   NEW_KEY,W       ; Get row of key pressed
```

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14          PAGE 11
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

0144 390F            0528         andlw   00Fh
0145 1903            0529         btfsc   STATUS,Z
0146 2955            0530         goto    DECOUT          ;If no key pressed, exit
0147 00AF            0531         movwf   TEMP1
0148 30FF            0532         movlw   0FFH
                     0533 DECL2
0149 3E01            0534         addlw   001h            ; Count row
014A 0CAF            0535         rrf     TEMP1           ; Rotate row until it is found
014B 1C03            0536         btfss   STATUS,C
014C 2949            0537         goto    DECL2
014D 00AF            0538         movwf   TEMP1
014E 1003            0539         bcf     STATUS,C
014F 0DAF            0540         rlf     TEMP1           ; Move row to upper nibble
0150 0DAF            0541         rlf     TEMP1
0151 082F            0542         movf    TEMP1,W
0152 072A            0543         addwf   TEMP,W          ; Add column to row value
0153 2156            0544         call    DEC_TABL        ; Get ASCII value of key
0154 00AC            0545         movwf   NEW_KEY
                     0546 DECOUT
0155 0008            0547         return
                     0548 DEC_TABL
0156 00AA            0549         movwf   TEMP            ; Save key value
0157 3001            0550         movlw   01h
0158 008A            0551         movwf   PCLATH          ; Setup for second page of RAM
0159 082A            0552         movf    TEMP,W
015A 0782            0553         addwf   PCL             ; Jump, return with ASCII value
015B 3446            0554         retlw   'F'
015C 3442            0555         retlw   'B'
015D 3430            0556         retlw   '0'
015E 3441            0557         retlw   'A'
015F 3445            0558         retlw   'E'
0160 3439            0559         retlw   '9'
0161 3438            0560         retlw   '8'
0162 3437            0561         retlw   '7'
0163 3444            0562         retlw   'D'
0164 3436            0563         retlw   '6'
0165 3435            0564         retlw   '5'
0166 3434            0565         retlw   '4'
0167 3443            0566         retlw   'C'
0168 3433            0567         retlw   '3'
0169 3432            0568         retlw   '2'
016A 3431            0569         retlw   '1'
                     0570
                     0571 ;*******************************************************************
                     0572 ;*This routine is a software delay.                                 *
                     0573 ;*At 4Mhz clock, the loop takes 3uS, so initialize TEMP with      *
                     0574 ;*a value of 3 to give 9uS, plus the move etc should result in   *
                     0575 ;*a total time of > 10uS.                                        *
                     0576 ;*******************************************************************
                     0577
                     0578 SetupDelay
016B 0BAA            0579         decfsz  TEMP
016C 296B            0580         goto    SetupDelay
```


© 1994 Microchip Technology Inc.

```
MPASM 01.00.02 Alpha  SCAPPNOT\IIC   5-10-1994  13:2:14                PAGE 12
 Demonstration of I2C MultiMaster mode

LOC  OBJECT CODE     LINE SOURCE TEXT

016D 0008            0581         return
                     0582
                     0583         END
                     0584


MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : X-X------ XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX


0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

SECTION 4
PIC17CXX APPLICATION NOTES

4

Saving and Restoring Status on Interrupt - AN534 ..... 4- 1
Implementing Table Read and Table Write - AN548 ..... 4- 3
Frequency and Resolution Options for PWM Outputs - AN539 ..... 4- 7
Using PWM to Generate Analog Output - AN538 ..... 4- 15
Using the PWM - AN564 ..... 4- 17
Using the Capture Module - AN545 ..... 4- 29
Serial Port Utilities - AN547 ..... 4- 53
Math Utility Routines - AN544 ..... 4- 63
Implementing IIR Digital Filters - AN540 4-129
Implementation of Fast Fourier Transforms - AN542 4-145
Tone Generation - AN543 4-163
Servo Control of a DC Brush Motor - AN532 4-185
Implementation of the Data Encryption Standard Using PIC17C42 - AN583 4-273

© 1994 Microchip Technology Inc.

Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)

INTRODUCTION

The PIC17C42 has a 16 level deep hardware stack. The program counter is pushed into this stack on interrupts and subroutine calls. However, other key registers are not saved in the stack. Registers such as W, ALUSTA (which has carry, zero and other flag bits) and the bank select register (BSR) must be saved in an interrupt service routine. The following macros, PUSH and POP implement a parameter stack in data memory to accomplish this.

The indirect addressing register FSR0 is used to implement this parameter stack. It is assumed that FSR0 and its control bits are not used or modified elsewhere. The stack pointer (FSR0) is initialized at the highest RAM location (FFh).


```
Main_prog       SETF    FSR0            ;Initialize and dedicate FSR0 as stack pointer
                BCF     ALUSTA,FS0      ;
                BCF     ALUSTA,FS1      ;Set-up FSR0 for auto-dec
                .
                .
                .
PUSH            MACRO
                BCF     ALUSTA,FS0      ;Set-up FSR0 for auto-dec
                MOVFP   ALUSTA,IND0     ;Save ALUSTA first
                MOVFP   BSR,IND0        ;
                MOVFP   W,IND0          ;
                MOVFP   RAM_x, IND0     ;Now save general
                MOVFP   RAM_y, IND0     ;Purpose registers
                ENDM
POP             MACRO
                BSF     ALUSTA,FS0      ;Set-up for auto-inc
                INCF    FSR0            ;
                MOVPF   IND0, RAM_y     ;
                MOVPF   IND0, RAM_x     ;
                MOVPF   IND0,W          ;
                MOVPF   IND0,BSR        ;
                MOVFP   IND0,ALUSTA     ;restore ALUSTA last
                DECF    FSR0            ;Adjust stack pointer
                ENDM
                .
                .
                .
interrupt_routine
                PUSH                    ;save registers
                .
                .
                .
;main body of interrupt service
                .
                .
                .
                POP                     ;restore status
                RETFIE                  ;return
```

© 1993 Microchip Technology Inc.

Saving and Restoring Status on Interrupt

1. MOVFP instruction does not affect status flags while MOVPF does.
2. MOVFP and MOVPF are used such that any register can be saved and restored. Note that register being saved or restored has address f (which can be 00h to FFh) and other address, IND0 (indirect), therefore, can be any address.
3. FSR auto-increments or auto-decrements after the operation ('post'). Therefore, in the POP macro pre-increment is simulated.
4. All interrupts should be disabled when executing the PUSH and POP instructions. While PUSH will have the GIE bit disabled, POP may not necessarily have the GIE bit disabled. The user should disable the GIE bit when executing the POP.

Using this scheme, interrupts and subroutine calls can be nested, since the stack will grow and shrink. The stack can be used to pass parameters to subroutines.

Author: Stan D'Souza
Logic Products Division

© 1993 Microchip Technology Inc.

MICROCHIP

AN548

Implementing Table Read and Table Write

INTRODUCTION

This application brief discusses how to read data from program memory to data memory and write data from data memory to program memory.

RETLW K Instruction

As in all PIC17CXX family parts, the simplest method used to retrieve data from program memory to data memory is to use the RETLW K instruction. For example:

```
; simple program to transfer
; table values to PortB
Main
      movlw  5,W          ;load offset
      call   SimpleTableRead
      movwf  PortB        ;output to PortB
      •
      •
      •
SimpleTableRead
      addwf  PC           ;add offset to PC
Table retlw  0            ;return a known
                          ;table value based
                          ;on the OFFSET.
      •
      •
      •
      retlw  10
```

In the example above, OFFSET is loaded with the required offset to the Table and the subroutine SimpleTableRead is called. The table value is returned in the W register. In this manner program memory can be transferred to data memory.

Table Read Instruction

The PIC17C42 has an expanded instruction set which includes the TABLRD and TLRD instructions. These instructions are specifically constructed to transfer data from program memory to data memory.

If the instruction syntax is: TABLRD t,i,f, the sequence in which this instruction is executed is as follows:

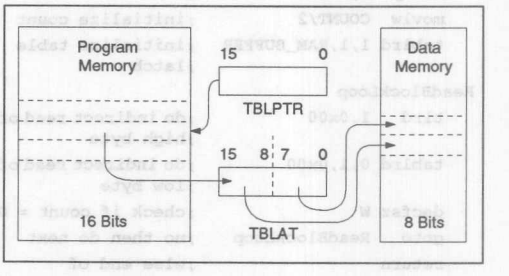
- if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
- else (if t = 0) the low byte of the table latch (TBLATL) is loaded in the file register f.
- next, the 16 bit data pointed to by the table pointer (TBLPTR) is loaded into the table latch.
- lastly, if i = 1 the table pointer (TBLPTR) is incremented.

Note: The first time this instruction is executed in a sequence, the table latch will not be initialized, hence an unknown value will be loaded in the file register. This is not a problem if the user overwrites the same f register in the next subsequent instruction.

If the instruction syntax is: TLRD t,f, the sequence in which this instruction is executed is as follows:

- if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
- else (if t = 0) the low byte of the table latch (TBLATH) is loaded in the file register f.

FIGURE 1 - TABLE READ



4

© 1993 Microchip Technology Inc.

Implementing Table Read and Write in PIC17C42

Read In Line

A simple method of transferring data from program memory to data memory is to use the TABLRD and TLRD instruction in sequence as shown in the example below:

```
;transfer 6 bytes of data in program memory at
;0x500, to data memory at 0x80:

ReadInLine
   movlw 05          ;load table pointer with
                     ; 0x500
   movwf TBLPTRH     ;       /
   clrf  TBLPTRL     ;       /
   tablrd0,1,0x80    ;get 16 bit value in
                     ;table latch.
   tlrd  0,0x80      ;low byte (1st) @ 80
   tablrd1,1,0x81    ;high byte (2nd) @ 81
   tlrd  0,0x82      ;3rd byte @ 82
   tablrd1,1,0x83    ;4th byte @ 83
   tlrd  0,0x84      ;5th byte @ 84
   tablrd1,1,0x85    ;6th byte @ 85
```

Reading a Block of Data

In instances where a block of N bytes needs to be transferred from program memory to data memory, the TABLRD and TLRD instruction need to be included in a loop which checks for N transfers.

```
;transfer 'COUNT' bytes (even values only) of
;data at program memory 'MESSAGE' to data memory
;at:
;'RAM_BUFFER'
ReadBlock
   movlw  high MESSAGE    ;load table pointer
   movpf  W,TBLPTRH       ;       /
   movlw  low MESSAGE     ;       /
   movpf  W,TBLPTRL       ;       /
   bcf    ALUSTA,5        ;enable post auto
                          ;increment of FSR0
   movlw  RAM_BUFFER      ;initialize FSR0 to
                          ;RAM_BUFFER
   movfp  W,FSR0          ;       /
   movlw  COUNT/2         ;initialize count
   tablrd 1,1,RAM_BUFFER  ;initialize table
                          ;latch
ReadBlockLoop
   tlrd   1,0x00          ;do indirect read of
                          ;high byte
   tablrd 0,1,0x00        ;do indirect read of
                          ;low byte
   decfsz W               ;check if count = 0
   goto   ReadBlockLoop   ;no then do next
   return                 ;else end of
                          ;transfer.
```

Program	Code Size	Transfer Rate
Simple Table Read (using RETLW)	N + 3	6 cycles/byte
Read In-Line	4 + N + N/2	1.5 cycles/byte
Read Block (using loop)	14 + N/2	3 cycles/byte

N = Number of bytes to transfer

	Code size		
	Simple Table Read	Read In-Line	Read Block
N = 10	13	19	19
N = 20	23	34	24

Conclusion:

In cases where the number of bytes to be transferred is small, the Read In-Line offers small code size for fast transfer rate. However, as the number of bytes to be transferred increases, the Read Block offers optimum code size for a decent transfer rate.

© 1993 Microchip Technology Inc.

Implementing Table Read and Write in PIC17C42

Table Write Instruction

The PIC17C42 has a TABLWT and a TLWT instruction which transfer data from data memory to program memory. Note in cases where the table pointer points to internal EPROM, the table write instruction will try to program the EPROM, hence the programming voltage must be present on the VPP line to successfully program the part.

The instruction syntax is: TABLWT t,i,f.

The sequence in which this instruction is executed is as follows:

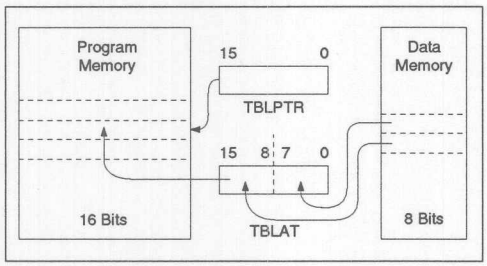
- if t= 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
- else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).
- next, the 16 bit data in the table latch is transferred to the program memory pointed to by the table pointer (TBLPTR).
- lastly, if i= 1 the table pointer (TBLPTR) is incremented.

The instruction syntax is: TLWT t,f

The sequence in which this instruction is executed is as follows:

- if t= 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
- else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).

FIGURE 2 - TABLE WRITE



Write in Line

A simple method of transferring data from data memory to program memory is to use the TABLWT and TLWT instruction in sequence as shown in the example below:

```
;transfer 6 bytes of data in data memory at 0x80,
to ;program memory at 0x5000:
ReadInLine
   movlw  50          ;load table pointer with
                      ;0x5000
   movwf  TBLPTRH     ;          /
   clrf   TBLPTRL     ;          /
   tlwt   1,0x80      ;high byte @ table latch.
   tablwt 0,1,0x81    ;low byte @ table latch;
                      ;latch  @ prog. mem.
   tlwt   1,0x82      ;3rd and 4th byte @ prog.
                      ;mem.
   tablwt 0,1,0x83    ;          /
   tlwt   1,0x84      ;5th and 6th byte @ prog.
                      ;mem.
   tablwt 0,1,0x85    ;          /
```

Writing a Block of Data

In instances where a block of N bytes needs to be transferred from data memory to program memory, the tablwt and tlwt instruction need to be included in a loop which checks for N transfers.

```
;transfer 'COUNT' bytes (even values only) of
;data at program memory at 'RAM_BUFFER' to
;program memory ;at 'MESSAGE'
WriteBlock
   movlw  high MESSAGE   ;load table pointer
   movpf  W,TBLPTRH      ;          /
   movlw  low MESSAGE    ;          /
   movpf  W,TBLPTRL      ;          /
   bcf    ALUSTA,5       ;enable post auto
                         ;increment of FSR0
   movlw  RAM_BUFFER     ;initialize FSR0 to
                         ;RAM_BUFFER
   movfp  W,FSR0         ;           /
   movlw  COUNT/2        ;initialize count
WriteBlockLoop
   tlwt   1,0x00         ;high byte @ table
                         ;latch
   tablwt 0,1,0x00       ;low byte @ table
                         ;latch;
                         ;table latch @ prog.
                         ;mem.
   decfsz W              ;check if count = 0
   goto   WriteBlockLoop ;no then do next
   return                ;else end of transfer.
```

Author: Stan D'Souza
Logic Products Division

© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

MICROCHIP

AN539

Frequency and Resolution Options for PWM Outputs

INTRODUCTION

The PIC17C42 is equipped with two high frequency Pulse Width Modulation (PWM) outputs. In a pulse width modulated signal the period of the signal is (usually) kept fixed, while the duty cycle is varied. In this application note, we will discuss options in selecting their frequency and resolution.

This application brief assumes that internal clock is used for the time-base, which is typically the preferred set-up. Also, throughout this application brief, PWM1 output is used in examples, timer1 is assumed to be the time-base.

Definition of terms:

Period of a PWM output is the duration after which the PWM pattern will repeat itself.

Frequency of a PWM output is = 1/Period.

Resolution of a PWM output is the granularity with which the duty cycle can be modulated.

In the case of the PIC17C42, when using PWM1 with timer1 as time-base the:

PWM1 period = [(PR1) + 1] x 4tosc

PWM1 duty cycle = (DC1) x tosc

where PR1 = period register for timer1

DC1 = PW1DCH, PW2DCL concantenated (10-bit value)

tosc = oscillator period

At 16 MHz oscillator frequency, tosc = 62.5 ns. The user can control the frequency of the PWM output by altering the 'period' value of the time-base. For example, if period is chosen to be 100 tosc (PR1 = 18h), then PWM frequency is 1/(100 x 62.5) ns = 160 KHz. Note however that duty cycle resolution is a little less than 7-bits.

Useful and Common PWM Modes

While a variety of period values can be selected, the following modes would be most commonly used:

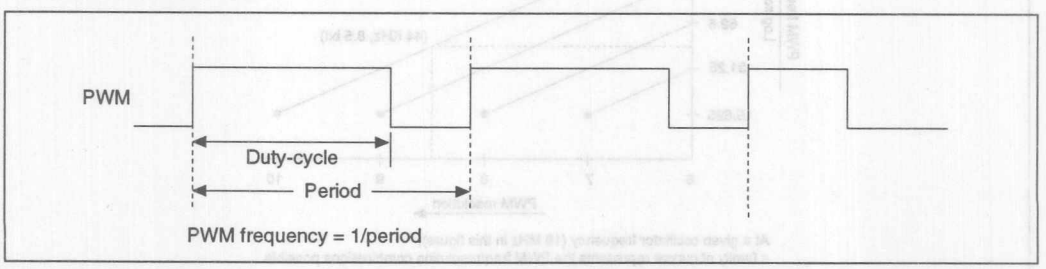
10-Bit Mode: In this mode PWM duty cycle has full 10-bit resolution (maximum offered by the PIC17C42). The period register PR1 is set at FFh. PWM period is 1024tosc = 64 µs. PWM frequency is 15.625 KHz. The user must write both PW1DCH and PW1DCL to update PWM output. See Appendix A for an example that code modules 10-bit resolution PWM output (PWM10.LST).

8-Bit Hi-Resolution Mode: In this mode, the user has only an 8-bit quantity to write to the duty-cycle register. Period register is set at 3Fh (63 decimal), such that PWM period is 256 tosc. To write the 8-bit duty-cycle value, first the 8-bit is right shifted two bits. The upper six bits are written to PW1DCH and the lower two bits are written to PW1DCL as follows:

```
;8-bit duty-cycle value is in W reg
CLRF  TEMP          ;
RRCF  WREG          ;
RRCF  TEMP          ;
RRCF  WREG          ;
RRCF  TEMP          ;Shift right twice
ANDLW 00111111b     ;Mask off two-high bits
MOVPF WREG,PW1DCH ;Write duty-cycle val-
ues
MOVFP TEMP,PW1DCL ;
```

Note that in 8-bit, hi-resolution mode, maximum PWM frequency is attained. For example, at 16 MHz clock, PWM period = 256 tosc = 16 µs; PWM frequency = 62.5 KHz. See appendix B for an example code that generates 8-bit low high resolution PWM output (PWM8HI.LST).

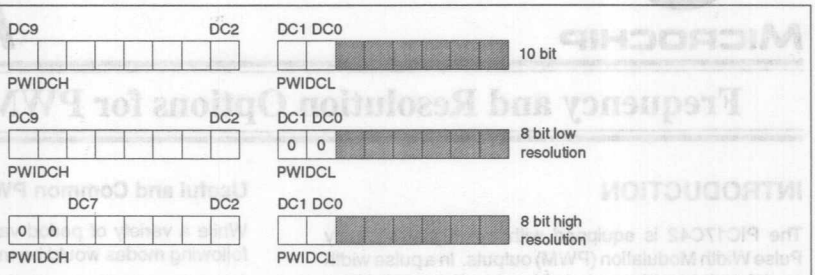
FIGURE 1 - PWM OUTPUT



© 1993 Microchip Technology Inc.

4

FIGURE 2 - VARIOUS PWM MODES



Note: DC registers suited for 16 bit computation; utilizing 10 MSB only.

8-Bit Low Resolution Mode

In this mode, the user still has only an 8-bit quantity to write to duty cycle register. However, the desired frequency of the PWM output is less, due to the nature of the application. For example, if the PWM output is being used to drive a motor through a power stage, the power transistors (or devices) due to their switching time will prefer PWM frequency not to exceed certain frequency. In the previous section, we derived an 8-bit resolution PWM output at 62.5 KHz.

To attain a low-resolution PWM output, the PW1DCL is always kept at zero. The 8-bit value is written to PW1DCH. The period (PR1) is set at FFh, i.e. 256 Tcy equals 1024 tosc (15.625 KHz). See Appendix C for an example code that produces 8-bit low resolution PWM output (PWM8LO.LST).

Choosing Resolution and Frequency of PWM Output

Actually, the resolution and the frequency of the PWM output is selectable within certain limits. The user will need to first define the requirements based on the application. There may be an upper limit to the frequency if the PWM is being used to drive motors. On the other hand, if the PWM is being filtered to generate an analog signal, higher frequency may be desirable. In any case, the lowest frequency achievable (using internal clock for the timer) is (OSC freq/1024). At 16 MHz oscillator input, the lowest PWM frequency possible is 15.625 KHz. At resolutions less than 10-bit, higher frequencies are possible (see Figure 3). For example, if 7-bit resolution is chosen, then the PWM frequencies can be 15-625 KHz, 31.25 KHz, 62.5 KHz or 125 KHz. The reader will note that it's how the 7-bits are placed within the 10-bit possible duty cycle value.

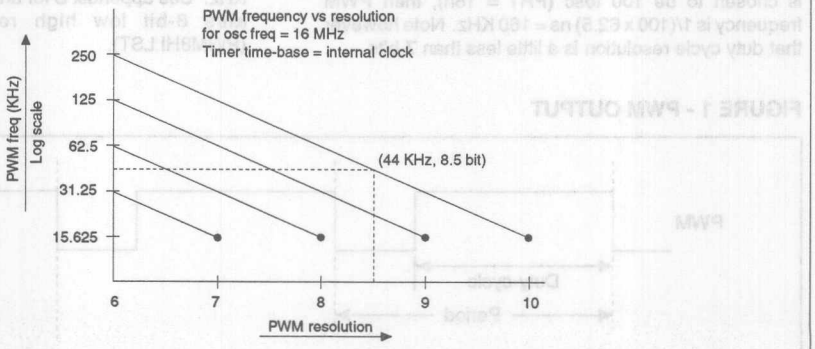
Conversely, if a certain frequency is desired, such as 44 KHz, then referring to Figure 1, resolution can be 8.5-bit or 7.5-bit or 6.5-bit etc.

Summary

The frequency and resolution of the PWM outputs of the PIC17C42 can be traded off against each other to best suit the application. The oscillator frequency can also be varied to adjust PWM frequency, if necessary. External clock should be used as timer time-base to generate very low frequency PWM output.

Authors: Stan D'Souza
Sumit Mitra
Logic Products Division

FIGURE 3 - PWM FREQ VS RESOLUTION



At a given oscillator frequency (16 MHz in this figure), a family of curves represents the PWM freq/resolution combinations possible.

© 1993 Microchip Technology Inc.

APPENDIX A: PWM10.LST

```
MPASM B0.54                                                    PAGE  1
PULSE WIDTH MODULATION 10 BIT RESOLUTION

                        TITLE   "PULSE WIDTH MODULATION 10 BIT RESOLUTION"
                        LIST    P=17C42, C=80, T=ON


                        include         "17c42.h"

0021            PWM_HI  equ     0x21
0020            PWM_LO  equ     0x20
0022            TEMP    equ     0x22

                ;The user would generate a 16 bit value which is saved in r
                ;locations PWM_HI and PWM_LO byte. In 10 bit mode, the prog
                ;transfers these values directly to the Duty Cycle (DC) reg
                ;generate the required 10 bit PWM.
                ;
                ;
                ;This is a short program to demonstrate how to generate PWM
                ;10 bit resolution. Since a 10Mhz crystal was used in the t
                ;The max. period = 1024x100nS = 102.4 uS or 9.8 KHz. This p
                ;keeps the period constant and varies the duty cycle (which
                ;to the most significant 10 bits of the 16 bit value PWM_LO
                ;This program is interrupt driven, i.e. the update to the D
                ;is done in the rtcc interrupt, which then enables the pwm
                ;The period update is done during the pwm interrupt. The pw
                ;ramps up from 0% to 100% duty cycle and then repeats. The
                ;sweep takes approx. 52 secs.
                ;
                ;
                        ORG     0
0000 C058               goto    start
                ;
                        ORG     0x10            ;vector for rtcc interrupt
                rtcc_int
0010 C04C               goto    service_rtcc    ;service rtcc
                ;
                        ORG     0x0020          ;vector for pwm interrupt
                pwm_int
0020 C03E               goto    service_pwm     ;service pwm only
                ;
                        ORG     0x0030
                ;
                ;initialize internal hardware to generate the output
                ;for 10 bit resolution pwm.
                init_pwm10
0030 B802               movlb   2
0031 2910               clrf    tmr1            ;clear timer 1
                                                ;used to "drive" pwm1
0032 2B14               setf    pr1             ;set period=9.8 khz
0033 B803               movlb   3
0034 7221               movfp   PWM_HI,pw1dch   ;load duty cyl. hi byte
0035 7020               movfp   PWM_LO,pw1dcl   ;load duty cycle lo byte
0036 2916               clrf    tcon1           ;tmr1 inc. internally
                                                ;as 8 bit counter
0037 B01B               movlw   00010001B       ;start tmr1 and
0038 4A17               movpf   wreg,tcon2      ;enable pwm1
0039 B801               movlb   1
003A 2917               clrf    pie             ;clr all int. enables
003B 2916               clrf    pir             ;clear all interrupts
003C 8307               bsf     _peie           ;except peripheral int.
003D 0005               retfie
                ;
                ;
```

© 1993 Microchip Technology Inc.

```
                        ;everytime a new value is written to the PWM_HI, PWM_LO reg
                        ;tmr1 interrupts is enabled. The DC value are written just
                        ;the "pwm interrupt" is enabled. Here the new period regist
                        ;updated. In this example, period is kept constant at 0xff
                        service_pwm
                                ;if the period changed, write new value here.
003E B802                       movlb   2               ;select bank 2
003F 2B14                       setf    pr1             ;period <- 0xff
0040 B801                       movlb   1               ;disable tmr1 int
0041 8C17                       bcf     _tm1ie          ;       /
0042 0005                       retfie
                        ;
                        ;This part of the program is basically used to simmulate a
                        ;which would be used to drive the pwm output.
                        ;
                        ;the rtcc is set up to interrupt every 52 mS.
                        init_rtcc
0043 B00B                       movlw   00100000b       ;set up rtcc timer
0044 650A                       movfp   wreg,rtcsta     ;       /
0045 290B                       clrf    rtccl           ;clear rtcc
0046 290C                       clrf    rtcch           ;       /
0047 B080                       movlw   0x80            ;start pwm at 50%
0048 0121                       movwf   PWM_HI          ;       /
0049 2920                       clrf    PWM_LO          ;       /
004A 8107                       bsf     _rtcie          ;enable rtcc int.
004B 0002                       return
                        ;
                        ;Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
                        ;Only the 10 most significant bits are incremented. Once th
                        ;
                        service_rtcc
004C 8D07                       bcf     _rtcir          ;reset int flag
                                ;do a pseudo inc of the 10 bit PWM_HI, PWM_LO.
004D 8804                       bcf     _carry          ;clear carry
004E B00B                       movlw   01000000b       ;load lsb for 10 bit
004F 0F20                       addwf   PWM_LO,1        ;add to LSB
0050 9804                       btfsc   _carry          ;carry?
0051 1521                       incf    PWM_HI          ;yes then inc PWM_HI
                                ;now load the values into the Duty Cycle registers

0052 B803                       movlb   3               ;bank 3
0053 7020                       movfp   PWM_LO,pw1dcl   ;load lo value
0054 7221                       movfp   PWM_HI,pw1dch   ;load hi value
0055 B801                       movlb   1
0056 8417                       bsf     _tm1ie          ;enable tmr1 int
0057 0005                       retfie
                        ;
                        ;
                        start
0058 8406                       bsf     _glintd         ;disable interrupts
0059 E043                       call    init_rtcc       ;initailize the RTCC tmr
                                                        ;for test purposes
005A E030                       call    init_pwm10      ;initialize pwm
005B C05B               loop    goto    loop            ;spin wheels
                        ;
                                END


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

Appendix B: PWM8HI.LST

```
MPASM B0.54                                                    PAGE  1
PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION


                        TITLE   "PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION"
                        LIST            P=17C42, C=80, T=ON

                        include         "17c42.h"


0021                PWM_HI  equ     0x21
0020                PWM_LO  equ     0x20
0022                TEMP    equ     0x22
                    ;The user would generate a 16 bit value which is saved in r
                    ;locations PWM_HI and PWM_LO byte. In 8 bit hi-res mode, th
                    ;transfers the 8 bit values to the lo Duty Cycle (DC) regis
                    ;generate the required 8 bit hi-res PWM.
                    ;
                    ;
                    ;This is a short program to demonstrate how to generate PWM
                    ;8 bit resolution. Since a 10Mhz crystal was used in the te
                    ;The max. period = 256x100nS = 25.6uS or 39KHz. This progra
                    ;keeps the period constant and varies the duty cycle (which
                    ;to the most significant 10 bits of the 16 bit value PWM_LO
                    ;This program is interrupt driven, i.e. the update to the D
                    ;is done in the rtcc interrupt, which then enables the pwm
                    ;The period update is done during the pwm interrupt. The pw
                    ;ramps up from 0% to 100% duty cycle and then repeats. The
                    ;sweep takes approx. 13.3 secs.
                    ;
                    ;
                            ORG     0
0000 C063                   goto    start
                    ;
                            ORG     0x10            ;vector for rtcc interrupt

                    rtcc_int
0010 C054                   goto    service_rtcc    ;service rtcc
                    ;
                            ORG     0x0020          ;vector for pwm interrupt
                    pwm_int
0020 C046                   goto    service_pwm     ;service pwm only
                    ;
                            ORG     0x0030
                    ;
                    ;initialize internal hardware to generate the pwm output
                    init_pwm8hi
0030 B802                   movlb   2
0031 2910                   clrf    tmr1            ;clear timer 1
                                                    ;used to "drive" pwm1
0032 B062                   movlw   62              ;set period=39khz
0033 0114                   movwf   pr1             ;       /
0034 B803                   movlb   3
0035 2922                   clrf    TEMP            ;TEMP = mask for pw1dcl
0036 6A21                   movfp   PWM_HI,wreg     ;get duty cyl. hi byte
0037 190A                   rrcf    wreg            ;rotate hi through carry
0038 1922                   rrcf    TEMP            ;rotate into lo byte
0039 190A                   rrcf    wreg            ;repeat for 2nd lsb
003A 1922                   rrcf    TEMP            ;       /
003B B53F                   andlw   b'00111111'     ;mask hi bits
003C 4012                   movpf   W,pw1dch        ;save in high
003D 7022                   movfp   TEMP,pw1dcl     ;save in low
003E 2916                   clrf    tcon1           ;tmr1 inc. internally
                                                    ;as 8 bit counter
003F B011                   movlw   b'00010001'     ;start tmr1 and
0040 4017                   movpf   W,tcon2         ;enable pwm1
0041 B801                   movlb   1
```

© 1993 Microchip Technology Inc.

```
0042 2917                       clrf    pie             ;clr all int. enables
0043 2916                       clrf    pir             ;clear all interrupts
0044 8307                       bsf     _peie           ;except peripheral int.
0045 0005                       retfie
                        ;
                        ;
                        ;everytime a new value is written to the PWM_HI, PWM_LO reg
                        ;tmr1 interrupts is enabled. The DC value are written just
                        ;the "pwm interrupt" is enabled. Here the new period regist
                        ;updated. In this example, period is kept constant at 62 Tc
                        service_pwm
                               ;if the period changed, write new value here.
0046 B802                       movlb   2               ;select bank 2
0047 B062                       movlw   62              ;period = 62 Tcyl.
0048 0114                       movwf   pr1             ;       /
0049 B801                       movlb   1               ;disable tmr1 int
004A 8C17                       bcf     _tm1ie          ;       /
004B 0005                       retfie
                        ;
                        ;This part of the program is basically used to simmulate a

                        ;which would be used to drive the pwm output.
                        ;
                        ;the rtcc is set up to interrupt every 52 mS.
                        init_rtcc
004C B020                       movlw   b'00100000'     ;set up rtcc timer
004D 650A                       movfp   wreg,rtcsta     ;       /
004E 290B                       clrf    rtccl           ;clear rtcc
004F 290C                       clrf    rtcch           ;       /
0050 B031                       movlw   31              ;init pwm at 50%
0051 0121                       movwf   PWM_HI          ;save in high
0052 8107                       bsf     _rtcie          ;enable rtcc int.
0053 0002                       return
                        ;
                        ;Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen

                        ;Only the 8 most significant bits are incremented.
                        ;
                        service_rtcc
0054 8D07                       bcf     _rtcir          ;reset int flag
                                                        ;do a pseudo inc of the 8 bit PWM_HI.
0055 1521                       incf    PWM_HI          ;inc PWM_HI
                                                        ;now load the values into the Duty Cycle

0056 B803                       movlb   3               ;bank 3
0057 2922                       clrf    TEMP            ;TEMP = mask for pw1dcl
0058 6A21                       movfp   PWM_HI,wreg     ;get duty cyl. hi byte
0059 190A                       rrcf    wreg            ;rotate hi through carry
005A 1922                       rrcf    TEMP            ;rotate into lo byte
005B 190A                       rrcf    wreg            ;repeat for 2nd lsb
005C 1922                       rrcf    TEMP            ;       /
005D B53F                       andlw   b'00111111'     ;mask hi bits
005E 4A12                       movpf   wreg,pw1dch     ;save in high
005F 7022                       movfp   TEMP,pw1dcl     ;save in low
0060 B801                       movlb   1
0061 8417                       bsf     _tm1ie          ;enable tmr1 int
0062 0005                       retfie
                        ;
                        ;
                        start
0063 8406                       bsf     _glintd         ;disable interrupts
0064 E04C                       call    init_rtcc       ;initailize the RTCC tmr
                                                        ;for test purposes
0065 E030                       call    init_pwm8hi     ;initialize 8 bit pwm
0066 C066               loop    goto    loop            ;spin wheels.
                        ;
                               END

Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

Appendix C: PWM8LO.LST

```
MPASM B0.54                                                    PAGE  1
PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION

                        TITLE   "PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION"
                        LIST    P=17C42, T=ON, C=80

                    include "17c42.h"

0021                PWM_HI  equ     0x21
0020                PWM_LO  equ     0x20
0022                TEMP    equ     0x22
                    ;The user would generate a 16 bit value which is saved in r
                    ;locations PWM_HI and PWM_LO byte. In 8 bit lo-res mode, th
                    ;transfers the 8 hi-byte value directly to the PW1DCH regis
                    ;
                    ;
                    ;This is a short program to demonstrate how to generate PWM
                    ;8 bit low resolution. Since a 5.068Mhz crystal was used in
                    ;The max. period = 1024x100nS = 102.4 uS or 9.8 KHz. This p
                    ;keeps the period constant and varies the duty cycle (which
                    ;to the most significant 8 bits of the 16 bit value PWM_LO&
                    ;This program is interrupt driven, i.e. the update to the D
                    ;is done in the rtcc interrupt, which then enables the pwm
                    ;The period update is done during the pwm interrupt. The pw
                    ;ramps up from 0% to 100% duty cycle and then repeats. The
                    ;sweep takes approx. 52 secs.
                    ;
                    ;
                            ORG     0
0000 C053                   goto    start
                    ;
                            ORG     0x10            ;vector for rtcc interrupt
                    rtcc_int
0010 C04C                   goto    service_rtcc    ;service rtcc
                    ;
                            ORG     0x0020          ;vector for pwm interrupt
                    pwm_int
0020 C03E                   goto    service_pwm     ;service pwm only
                    ;
                            ORG     0x0030
                    ;
                    ;initialize internal hardware to generate the output
                    ;for 8 bit low resolution pwm
                    init_pwm8lo
0030 B802                   movlb   2
0031 2910                   clrf    tmr1            ;clear timer 1
                                                    ;used to "drive" pwm1
0032 2B14                   setf    pr1             ;set period=9.8 khz
0033 B803                   movlb   3
0034 7221                   movfp   PWM_HI,pw1dch   ;load duty cyl. hi byte
0035 2910                   clrf    pw1dcl          ;clear lo byte
0036 2916                   clrf    tcon1           ;tmr1 inc. internally
                                                    ;as 8 bit counter
0037 B011                   movlw   b'00010001'     ;start tmr1 and
0038 4A17                   movpf   wreg,tcon2      ;enable pwm1
0039 B801                   movlb   1
003A 2917                   clrf    pie             ;clr all int. enables
003B 2916                   clrf    pir             ;clear all interrupts
003C 8307                   bsf     _peie           ;except peripheral int.
003D 0005                   retfie
                    ;
                    ;
```

© 1993 Microchip Technology Inc.

```
                        ;everytime a new value is written to the PWM_HI, PWM_LO reg
                        ;tmr1 interrupts is enabled. The DC value are written just
                        ;the "pwm interrupt" is enabled. Here the new period regist
                        ;updated. In this example, period is kept constant at 0xff
                        service_pwm
                              ;if the period changed, write new value here.
003E B802                     movlb   2               ;select bank 2
003F 2B14                     setf    pr1             ;period <- 0xff
0040 B801                     movlb   1               ;disable tmr1 int
0041 8C17                     bcf     _tm1ie          ;       /
0042 0005                     retfie
                        ;
                        ;This part of the program is basically used to simmulate a
                        ;which would be used to drive the pwm output.
                        ;
                        ;the rtcc is set up to interrupt every 52 mS.
                        init_rtcc
0043 B020                     movlw   b'00100000'     ;set up rtcc timer
0044 6500                     movfp   W,rtcsta        ;       /
0045 290B                     clrf    rtccl           ;clear rtcc
0046 290C                     clrf    rtcch           ;       /
0047 B080                     movlw   0x80            ;begin PWM at 50% dc
0048 0121                     movwf   PWM_HI          ;       /
0049 2920                     clrf    PWM_LO          ;       /
004A 8107                     bsf     _rtcie          ;enable rtcc int.
004B 0002                     return
                        ;
                        ;Every rtcc interrupt, the PWM_HI&PWM_LO bytes are incremen
                        ;Only the 8 most significant bits are incremented.
                        ;
                        service_rtcc
004C 8D07                     bcf     _rtcir          ;reset int flag
                                                      ;do a inc of the 8 bit PWM_HI
004D 1521                     incf    PWM_HI
                                                      ;now load the values into the Duty Cycle
004E B803                     movlb   3
004F 7221                     movfp   PWM_HI,pw1dch   ;load hi byte
0050 B801                     movlb   1
0051 8417                     bsf     _tm1ie          ;enable tmr1 int
0052 0005                     retfie
                        ;
                        ;
                        start
0053 8406                     bsf     _glintd         ;disable interrupts
0054 E043                     call    init_rtcc       ;initailize the RTCC tmr
                                                      ;for test purposes
0055 E030                     call    init_pwm8lo     ;initialize pwm
0056 C056               loop  goto    loop            ;spin wheels.
                        ;
                              END

Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

MICROCHIP

AN538

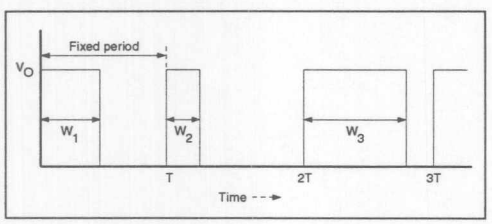
Using PWM to Generate Analog Output

Pulse Width Modulated (PWM) waveforms which are basically digital waveforms, can be used as cheap Digital-to-Analog (D/A) converters with few external components. A wide variety of microcontroller applications exists that need analog output but do not require high resolution D/A converters. Some speech applications (talk back units, speech synthesis systems in toys, etc.) also do not require high resolution D/A converters. For these applications, Pulse Width Modulated outputs may be used to convert to analog outputs.

Conversion of PWM waveforms to analog signals involves the use of analog low-pass filters. This brief application note describes the design criteria of the analog filters necessary and the requirements of the PWM frequency. Later in this application note, a simple RC low-pass filter is designed to convert pulse-width modulated speech signals of 4 KHz bandwidth.

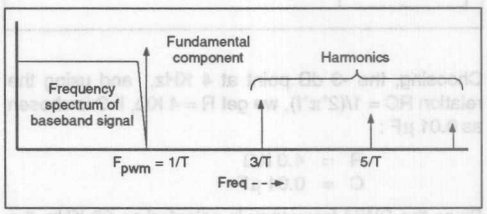
In a typical PWM signal, the base frequency is fixed, but the pulse width is a variable. The pulse width is directly proportional to the amplitude of the original unmodulated signal. In other words, in a PWM signal, the frequency of the waveform is a constant while the duty cycle varies (from 0 % to 100 %) according to the amplitude of the original signal. A typical PWM signal is shown in Figure 1.

FIGURE 1 - A TYPICAL PWM WAVEFORM



A Fourier analysis of a typical PWM signal (like the one depicted in Figure 1) shows that there is a strong peak at frequency $F_n = 1/T$. Other strong harmonics also exist at F = K/T, where K is an integer. These peaks are unwanted noise and should be eliminated. This requires that the PWM signal be low-pass filtered, thus eliminating these inherent noise components (see Figure 2).

FIGURE 2 - FREQUENCY SPECTRUM OF A PWM SIGNAL

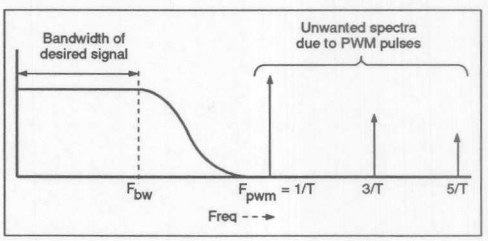


The band-width of the desired signal should thus be

$$f_{bw} <= (f_{pwm} = 1/T)$$

If f_{bw} is selected such that fbw=fpwm, then the external low-pass filter should be a brick-wall type filter. Brick-wall type analog filters are very difficult and expensive to build. So, for practical purpose, the external low-pass filter should be as shown in Figure 3.

FIGURE 3 - EXTERNAL LOW-PASS FILTER



This means, $f_{bw} << f_{pwm}$

or $f_{pwm} >> f_{bw}$

$=> f_{pwm} = K*f_{bw}$ (1)

where, K is a constant such that K >> 1

The value of K should be chosen depending upon by how many dB the inherent fundamental noise component of PWM be rejected. An example follows :

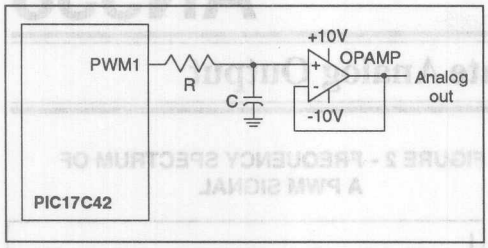
Example : It is required to design a simple RC low-pass filter to obtain an analog output from a pulse width modulated speech signal of bandwidth 4 KHz.

From eqn (1), choosing arbitrarily K = 5,

$f_{pwm} = K*f_{bw} = 5*4$ KHz = 20 KHz.

© 1993 Microchip Technology Inc.

FIGURE 4 - RC FILTER CONNECTED TO PWM1 OF PIC17C42



Choosing, the -3 dB point at 4 KHz, and using the relation RC = 1/(2*π*f), we get R = 4 KΩ, if C is chosen as 0.01 µF :

R = 4.0 KΩ
C = 0.01 µF

Since the PWM frequency is selected as 20 KHz, the fundamental noise peak to be filtered is at 20 KHz. Now, lets calculate by how many dB the main peak of PWM signal is cut-off at 20 KHz :

$$(dB)20KHz = -10*\log[1+ (2\pi f.RC)^2] = -14 \text{ dB}.$$

For many applications, this rejection of -14 dB will not suffice. Therefore instead of a simple RC low-pass filter, a higher order active low-pass filter may be necessary. Or if the microcontroller is capable of modulating at higher PWM frequencies, the rejection of noise will be more.

For example, using 8-bit resolution, the PIC17C42 can generate PWM frequency of 62.5 KHz.

At this frequency the attenuation of the PWM frequency is:

$$(dB)62.5KHz = -10*\log[1+ (2\pi f.RC)^2] = -24 \text{ dB}.$$

The higher frequency of the PIC17C42 PWM outputs makes it easier to generate analog output.

Author: Amar Palacherla
Logic Products Division

© 1993 Microchip Technology Inc.

Using the PWM

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of Pulse Width Modulation (PWM) using the PIC17C42 Timer1 or Timer2 modules. These examples may be modified to suit the specific needs of your application.

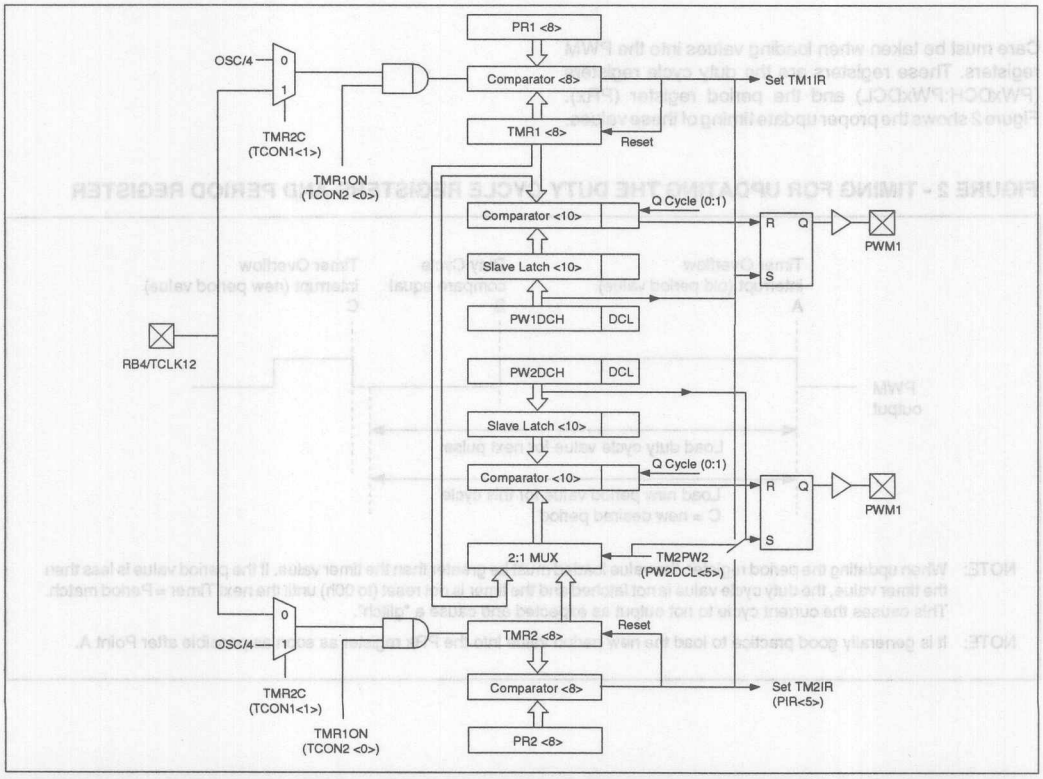
This Application Note describes the operation of the PWM. They include the following topics:

1. Simple PWM Operation
2. Variable Period / Variable Duty Cycle PWM
3 External Clock for Timer Timebase (ramifications/issues)

The listing file for the Variable Period / Variable Duty Cycle example can be found in Appendix A. The source files can be found on the Microchip BBS. On directions on how to access the Microchip BBS please refer to DS30128, which can also be found in the Microchip Embedded Control Handbook (Literature Number DS00092).

4

FIGURE 1 - TIMER1 AND TIMER2 BLOCK DIAGRAM WITH PWM MODE



© 1993 Microchip Technology Inc.

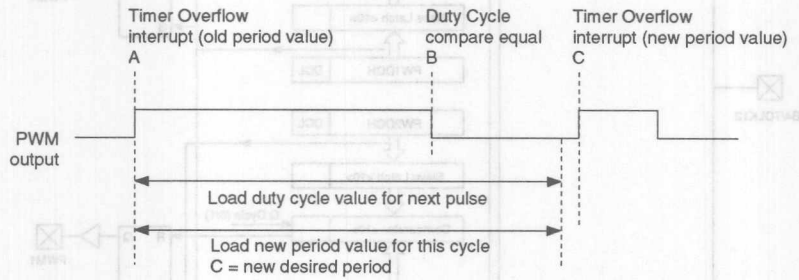
PWM Operation (cont.)

TABLE 1 - REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE

Name	BANK	ADDR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
INTSTA		0x06	PEIR	RTXIR	RTCIR	INTIR	PEIE	RTXIE	RTCIE	INTIE
CPUSTA		0x07	-	-	STKAV	GLINTD	$\overline{TO}$	$\overline{PD}$	-	-
TMR1	2	0x10	Timer1 Register							
TMR2	2	0x11	Timer2 Register							
PR1	2	0x14	Timer1 Period Register							
PR2	2	0x15	Timer2 Period Register							
PIR	1	0x16	IRB	TM3IR	TM2IR	TM1IR	CA2IR	CA1IR	TBMT	RBFL
PIE	1	0x17	IEB	TM3IE	TM2IE	TM1IE	CA2IE	CA1IE	TXIE	RCIE
PW1DCL	3	0x10	PWM1 Duty Cycle Low Register							
PW2DCL	3	0x11	PWM2 Duty Cycle Low Register							
PW1DCL	3	0x12	PWM1 Duty Cycle High Register							
PW2DCL	3	0x13	PWM2 Duty Cycle High Register							
TCON1	3	0x16	CA2ED1	CA2ED0	CA1ED1	CA1ED0	16/$\overline{8}$	TMR3C	TMR2C	TMR1C
TCON2	3	0x17	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/$\overline{PR3}$	TMR3ON	TMR2ON	TMR1ON

Care must be taken when loading values into the PWM registers. These registers are the duty cycle registers (PWxDCH:PWxDCL) and the period register (PRx). Figure 2 shows the proper update timing of these values.

FIGURE 2 - TIMING FOR UPDATING THE DUTY CYCLE REGISTERS AND PERIOD REGISTER



NOTE: When updating the period register, the value loaded must be greater than the timer value. If the period value is less then the timer value, the duty cycle value is not latched and the timer is not reset (to 00h) until the next Timer = Period match. This causes the current cycle to not output as expected and cause a "glitch".

NOTE: It is generally good practice to load the new period value into the PRx register as soon as possible after Point A.

© 1993 Microchip Technology Inc.

SIMPLE PWM OPERATION

Simple PWM operation is where the period of the PWM output remains constant, and only the duty cycle is modified. The PWM can operate in either of two modes:

- Hi-resolution mode-the PWxDCL register is modified
- Standard resolution mode - the PWxDCL register is not modified

When operating in the standard-resolution mode, only the PW-DCH register is ever modified. Since this takes only a single cycle, this can be done at any time. Also since the period is remaining constant this may be done without any PWM interrupt software overhead.

When operating in the high-resolution mode both the PWxDCH:PWxDCL register pair is modified. Since this is a multicycle update, care needs to be taken that the "new" PWM duty cycle value is not latched until the update is complete. If the duty cycle is latched before this update is complete, the duty cycle will display a "glitch". If the PWxDCH is written first, the maximum error is 3 Q-cycles (187.5 ns @ 16 MHz). If the PWxDCL is written first, the maximum error is also 3 Q-cycles (187.5 ns @ 16 MHz), with the PWxDCH delayed by one PWM period. This may be acceptable for some applications. If this is not acceptable for your application then a subroutine can be written to ensure that these duty cycle writes are not done when the timer will equal the period. One implementation of this subroutine (PWM_UD) is used in the Variable Period / Variable Duty Cycle PWM example. This is discussed in the following section, with the listing in Appendix A.

Additional code examples can be found in application note AN539 in the Embedded Control Handbook.

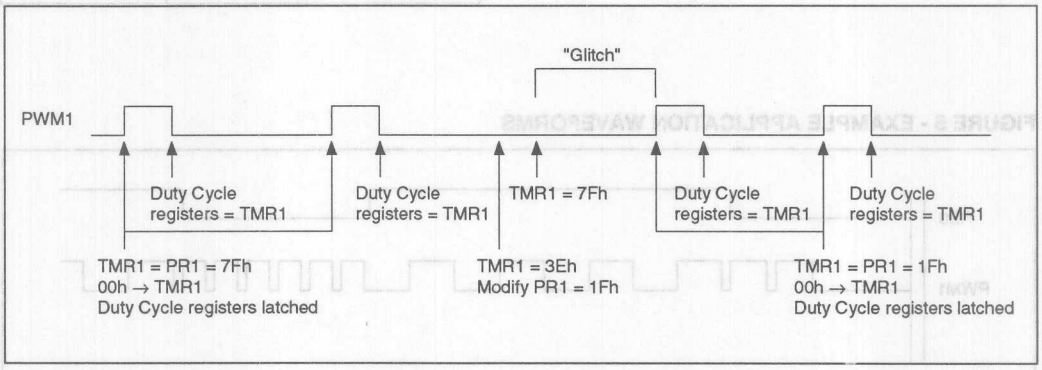
VARIABLE PERIOD / VARIABLE DUTY CYCLE PWM

In a variable period / variable duty cycle PWM both the duty cycle of the PWM as well as the frequency (period) of the PWM are modified.

The PIC17C42's hardware double buffers the duty cycle registers, but the period registers are not double buffered. What this means is that you can modify the duty cycle registers, but the value will only be latched when the timer register equals period register. Since the period register is not buffered, as the period register is modified this becomes the "new" period. This means that care must be taken when modifying the period register. The most common problem would be to modify the period register resulting in a "glitch" to occur. This "glitch" occurs when the period register is modified with a value that is less than the present timer value. The timer does not have a match with the old period value, and continues to count until the timer register equals period register.

Figure 3, shows an example where the period (PR1) register = 7Fh. Then the period is modified to a smaller value (PR1 = 1Fh) without checking that the value in Timer1 (TMR1) register = 3Eh. Since the new period (PR1) value is less then the present timer (TMR1) value, a glitch has occurred.

FIGURE 3 - MODIFYING PERIOD REGISTER "GLITCH"



© 1993 Microchip Technology Inc.

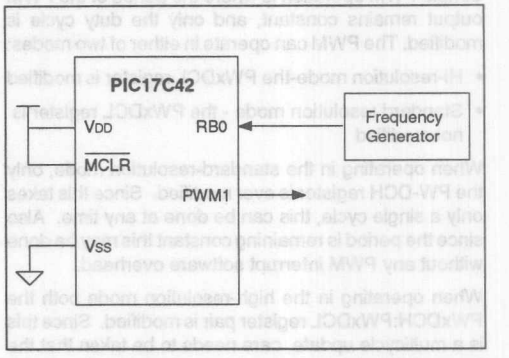
Care must be taken when writing a 10-bit duty cycle value. Since this requires two register writes, the Timer equals period could occur between these two writes, which would give a duty cycle that was not as expected. The cases are as follows:

1. If the duty cycle low (DCL) register is written, and then the Timer equals period. The old DCH register and the new DCL register becomes the duty cycle.
2. If the duty cycle high (DCH) register is written, and then the Timer equals period. The new DCH register and the old DCL register becomes the duty cycle

At the following occurrence of the timer equaling the period, the second register written would be updated. The subroutine PWM_UD (Appendix A) ensures that these duty cycle writes are not done when the timer will equal the period.

A software example of a variable period / variable duty cycle is shown in Appendix A. In this example the period is double buffered in software, and the new period value is loaded in the timer overflow interrupt service routine. When the new duty cycle needs to be loaded. The device connections are shown in Figure 4. This program has two PWM settings (period / duty cycle combinations) that are switched between depending on the level on pin RB0. A frequency generator was used to give a low frequency signal on the RB0 pin. Figure 5 shows an example of the input and output waveforms.

FIGURE 4 - APPLICATION HARDWARE SETUP

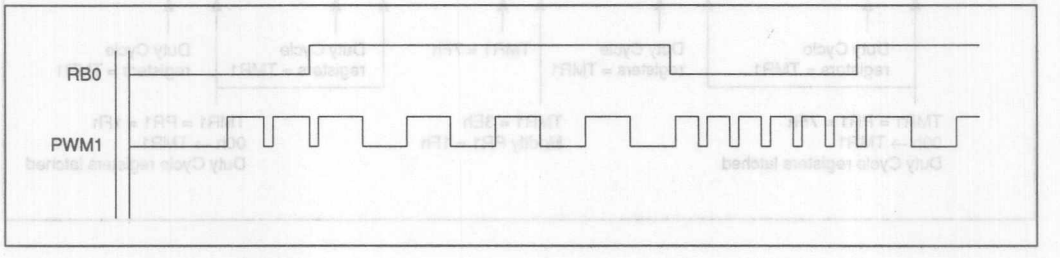


The program listing in Appendix A implements this example, Figure 8 is the flowchart of the program. This example may be modified to suit the particular needs of your application. The Table 3 is a summary of the requirements for this program (@ 16 MHz):

TABLE 3 - PROGRAM REQUIREMENTS

Code Size:	52 Words
RAM used:	11 Bytes
Interrupt Service Routine time	3.0 usec
Subroutine time	4.5 usec 6.0 usec
Maximum PWM frequency:	200 KHz
PWM Accuracy:	62.5 nsec

FIGURE 5 - EXAMPLE APPLICATION WAVEFORMS



© 1993 Microchip Technology Inc.

EXTERNAL CLOCK FOR TIMER TIMEBASE

The counters used for the time base of the PWM outputs can be software selected to operate from an external clock source. This allows a lower frequency PWM to be achieved. Doing this brings up new issues that must be understood for the application.

One of these issues is clock synchronization. All external clocks must be synchronized to the internal operating speed of the microcontroller, as shown in figure 9. When this synchronization occurs the PWM output is not truly operating from the external clock, but actually the internal synchronized clock. This leads to a "jitter" of the output to the clock. This jitter is caused from the delta time between the external clock and the synchronized clock not being constant. The synchronization errors are:

Duty cycle error = +/- Tcy

Period error = +/- Tcy

If you needed to run the PWM at a low frequency, and also want to reduce the "jitter" from the use of an external asynchronous clock, a PWM output could be used as the synchronous clock source. When the clock is synchronized to the device the clock error is always constant, so there is no jitter. Figure 7 shows this example.

FIGURE 7 - PWM OUTPUT TO GENERATE A SYNCHRONOUS CLOCK

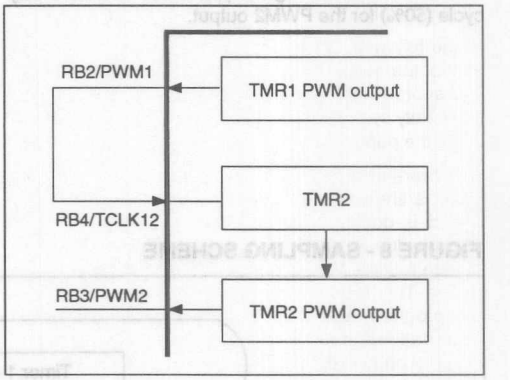
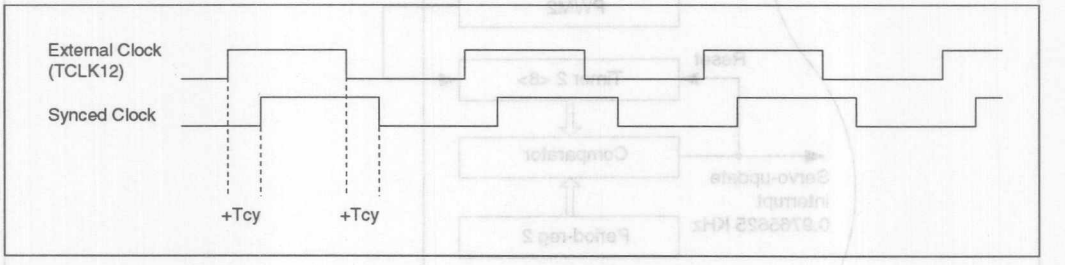


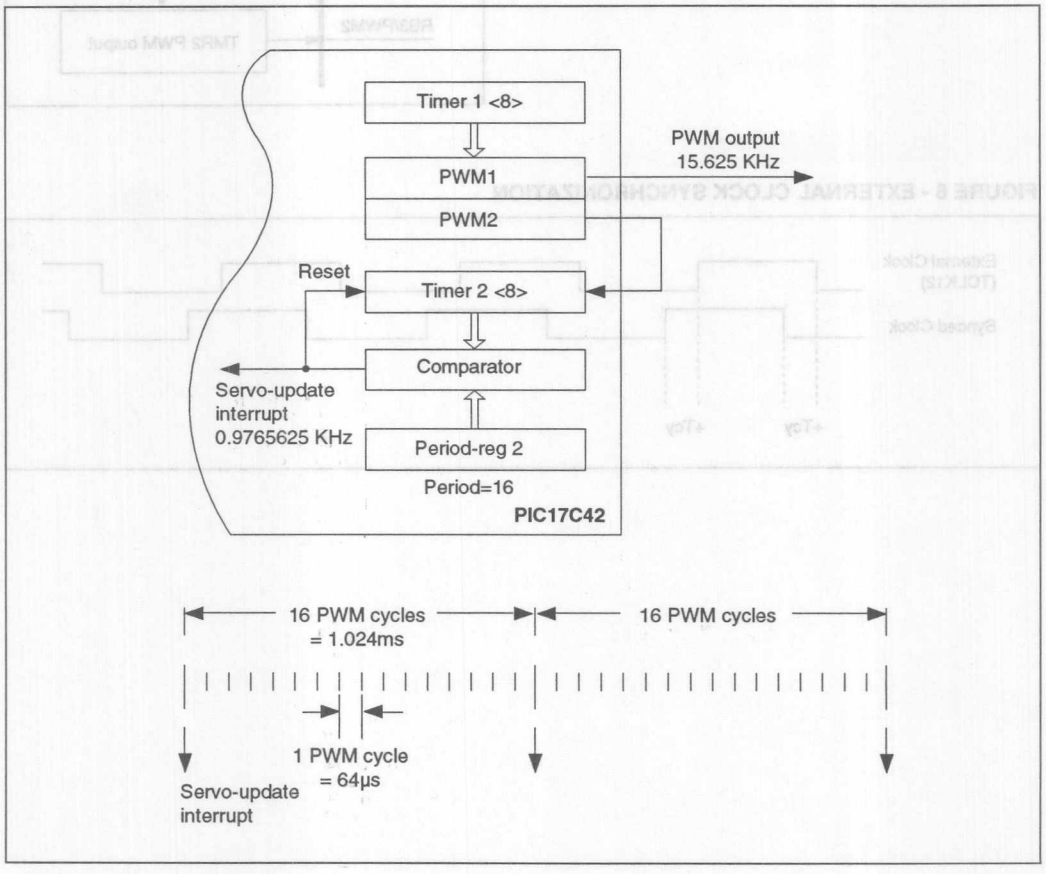
FIGURE 6 - EXTERNAL CLOCK SYNCHRONIZATION



Another use is where precise timing of updates need to be done, but not at the frequency of the PWM output. In this discussion, TMR1 is used as the time-base of a constant frequency PWM output. TMR1 uses the internal clock of the device and TMR2 uses the external clock input. TMR2 will get the clock input from the PWM2 output.

The PWM output is a constant frequency variable duty cycle output. The PW1DCH:PW1DCL register pair contain the variable duty cycle value of PWM1 output. The PW2DCH:PW2DCL register pair is set for a fixed duty cycle (50%) for the PWM2 output.

FIGURE 8 - SAMPLING SCHEME



The PWM outputs could be programmed to have a frequency of 20 KHz, so to reduce audible noise. The PWM2 signal is connected to the RB4/TCLK12, as shown in Figure 8. The PR2 register could be loaded with 14h (20), to give an interrupt every 1 KHz. This interrupt can then trigger tasks, such as updating the duty cycle of PWM1. This is useful in motor control as well as other applications where the update rate is less then the PWM frequency.

CONCLUSION

The PIC17C42's PWM features offer a high performance solution at a lower system cost then previously available. The versatility of PWMs make the PIC17C42 ideal for motor control applications (ses AN532) and many industrial control applications.

Author: Mark Palmer
Logic Products Division

© 1993 Microchip Technology Inc.

APPENDIX A: LISTING FILE

```
MPASM B0.54                                              PAGE  1


                    LIST   P=17C42, T=ON, C=120, N=0
                ; This is the basic outline for a program that generates a
                ; variable PWM output. The PWM's period and duty cycle can
                ; be varied. The new period (NEW_PR1) and the new duty cycle
                ; (NEW_DC1 and NEW_DC1Q) are loaded by the user program.
                ; The peripheral interrupt routine loads the new period value
                ; (frequency) into the PR1 register. A subroutine (PWM_UD)
                ; is also used to ensure that the 10-bit duty cycle registers
                ; are updated in the same PWM cycle, i.e. the timer match does not
                ; occur between two duty cycle register writes.
                ;
                ; The duty cycle value gets latched on the overflow (Period match)
                ; of the timer. The period value gets modified as soon as the period
                ; register is changed. Therefore care must be taken in updating
                ; the period register. In cases where the period value is modified
                ; to a smaller value, we must ensure that the Timer counter is less
                ; then this value when the period register is updated (TMR1 < new PR1).
                ; If TMR1 is greater then PR1, the counter will count to FFh, rollover
                ; to 00H, and only cause the overflow interrupt when it then reaches
                ; the period value. This would give a wrong PWM output.
                ;
                ; In this example the event which cause the PWM to be updated
                ; is an asynchronous event. A low frequency signal was placed on
                ; port pin RB0.
                ; For a high level the PWM registers are updated as follows:
                ;    PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
                ; For a low level the PWM registers are updated as follows:
                ;    PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
                ;
                ; Do the EQUate table
                ;
0020            NEW_DC1         EQU     0x20        ; New PWM1 duty cycle value
0021            NEW_DC1Q        EQU     0x21        ;
0022            NEW_PR1         EQU     0x22        ; New PWM1 period value
0025            PWM_WIN         EQU     0x25        ; Register for the PWM window cycle count
0026            CALC_PR         EQU     0x26        ; Calculated period value
0027            FLAG_REG        EQU     0x27        ; Register for flag bits
                ;
001A            DC1H            EQU     0x1A        ; PWM registers for high time
001B            DC1QH           EQU     0x1B
001C            PR1H            EQU     0x1C
                ;
001D            DC1L            EQU     0x1D        ; PWM registers for low time
001E            DC1QL           EQU     0x1E
001F            PR1L            EQU     0x1F
                ;
                ;
07FF            END_OF_PROG_MEM EQU 0x07FF
                ;
0004            ALUSTA          EQU     0x04
0006            CPUSTA          EQU     0x06
0007            INTSTA          EQU     0x07
000A            W               EQU     0x0A
                ;
0011            DDRB            EQU     0x11        ; Bank 0
0012            PORTB           EQU     0x12
                ;
0016            PIR             EQU     0x16        ; Bank 1
0017            PIE             EQU     0x17
                ;
```

© 1993 Microchip Technology Inc.

```
0010                    TMR1            EQU     0x10            ; Bank 2
0011                    TMR2            EQU     0x11
0012                    TMR31           EQU     0x12
0013                    TMR3h           EQU     0x13
0014                    PR1             EQU     0x14
0015                    PR2             EQU     0x15
0016                    PR3L            EQU     0x16
0017                    PR3h            EQU     0x17
                        ;
0010                    PW1DCL          EQU     0x10            ; Bank 3
0011                    PW2DCL          EQU     0x11
0012                    PW1DCH          EQU     0x12
0013                    PW2DCH          EQU     0x13
0016                    TCON1           EQU     0x16
0017                    TCON2           EQU     0x17

                                        ORG     0x0000          ; Origin for the RESET vector
0000 C02B                               GOTO    START           ; On reset, go to the start of
                                                                ;   the program
                                        ORG     0x0008          ; Origin for the external RA0/INT
                                                                ;   interrupt vector
0008 C07C                               GOTO    EXT_INT         ; Goto the ext. interrupt
                                                                ;   on RA0/INT routine
                                        ORG     0x0010          ; Origin for the RTCC
                                                                ;   overflow interrupt vector
0010 C07D                               GOTO    RTCCINT         ; Goto the RTCC overflow interrupt
                                                                ;   routine
                                        ORG     0x0018          ; Origin for the external
                                                                ;   RA1/RT interrupt vector
0018 C07E                               GOTO    RT_INT          ; Goto the ext. interrupt on
                                                                ;   RA1/RT routine
                                        ORG     0x0020          ; Origin for the interrupt vector
                                                                ;   of any enabled peripheral
                        ;
                        ; The interrupt routine for any peripheral interrupt, This routine
                        ; only deals with Timer1 interrupt.
                        ;
                        ; Time required to execute interrupt routine. Not including
                        ; interrupt latency (time to enter into the interrupt routine)
                        ;
                        ;       case1 - only T1 overflow            = 12 cycles
                        ;       case2 - Other                       = Infinite Loop
                        ;
                        ;
0020 B801               PER_INT         MOVLB   1               ; Select register Bank 1
0021 9416                               BTFSS   PIR,4           ; Did Timer1 overflow?
0022 C022               ERROR           GOTO    ERROR           ; Not a Timer1 overflow.
                                                ; No other interrupts should
                                                ; be enabled, so error.
                        ;
                        ; Once the enabled Timer1 overflow occurs, the period register
                        ; is loaded. This PWM waveform will remain until the PWM duty
                        ; cycle and / or period is updated. Until such update, there is no
                        ; S/W overhead from T1 interrupts (T1 interrupts can be disabled).
                        ;
                        ; NOTE: If PW1DCH >= PR1, then the duty cycle of this PWM output
                        ;       is 100%.
                        ;
                        ; NOTE: The new Period register (PR1) value, must always be greater
                        ;       than the value in the Timer1 register (TMR1). If a PR1 value
                        ;       is loaded that is less then the TMR1 value, the timer will
                        ;       continue to count until it reaches the PR1 value. I.E. TMR1
                        ;       will overflow at FFh and the count to the new PR1 value.
                        ;       Minimum PR1 value is 0Ah, due to time to load new values and
                        ;       execute the peripheral interrupt service routine.
                        ;
0023 8C16               T1OVFL          BCF     PIR,4           ; Clear Overflow interrupt flag
0024 B802                               MOVLB   2               ; Bank2
0025 7422                               MOVFP   NEW_PR1,PR1     ; Load this period value
```


© 1993 Microchip Technology Inc.

```
0026 B801                       MOVLB   1               ; Bank 0
0027 8C17                       BCF     PIE, 4          ; Disable T1 interrupt
                                                        ;   (until transition on PORTB0)
0028 B800                       MOVLB   0               ; Bank 0
0029 3F12                       BTG     PORTB, 7        ; Transition PortB 7 pin (H->L,
002A 0005                       RETFIE                  ; Return from Interrupt
                        ;
                        ; This is the start of the program.
                        ;
002B 8406               START   BSF     CPUSTA,4        ; Disable ALL interrupts via the
                                                        ;   Global Interrupt Disable
                                                        ;   (GLINTD) bit.
                                                        ;
                        MAIN                            ; Place Main program here
002C B803                       MOVLB   3               ; Select register Bank 3
002D 2817                       CLRF    TCON2,0         ; Stop the timers, Single Capture
002E B070                       MOVLW   0x070           ; Initalize TCON1 so that
002F 0116                       MOVWF   TCON1           ;   T1 (8-bit), T2 (8-bit),
                                                        ;   and T3 run off the internal
                                                        ;   system clock. Timer3 uses
                                                        ;   period register
0030 B00D                       MOVLW   0x0D            ; Load the PWM window cycle value
0031 0125                       MOVWF   PWM_WIN         ;
                        ;
0032 B800                       MOVLB   0               ; Select register Bank 0
0033 2B11                       SETF    DDRB, 1         ; Port B is an input
0034 2912                       CLRF    PORTB, 1        ; Set output values to 0 (for
0035 8F11                       BCF     DDRB, 7         ; PORTB7 is an output. Used to
0036 2927                       CLRF    FLAG_REG, 1     ; Clear the Flag registers
                        ;
                        ; Load registers with the PWM values that we will switch between. One set
                        ; for the time PORTB0 is high and another set for when low.
                        ;
                        ; For a high level the PWM registers are updated as follows:
                        ;    PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
                        ;    At 16Mhz this gives a period of 31.75 us, and a duty cycle of 16.625 us
                        ; For a low level the PWM registers are updated as follows:
                        ;    PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
                        ;    At 16Mhz this gives a period of 7.75 us, and a duty cycle of 6.00 us
                        ;
0037 B803                       MOVLB   3               ; Bank 3
0038 B03F                       MOVLW   0x3F            ; The Duty Cycle initial value is
0039 4A1A                       MOVPF   W, DC1H         ;   50% of the initial period
003A B040                       MOVLW   0x40            ;
003B 4A1B                       MOVPF   W, DC1QH        ; Duty Cycle low = 01
003C B007                       MOVLW   0x07            ; The Duty Cycle initial value is
003D 4A1D                       MOVPF   W, DC1L         ;   25% of the initial period
003E B080                       MOVLW   0x80            ;
003F 4A1E                       MOVPF   W, DC1QL        ; Duty Cycle low = 10
                        ;
0040 B802                       MOVLB   2               ; Bank 2
0041 B07F                       MOVLW   0x7F            ;
0042 4A1C                       MOVPF   W, PR1H         ; The initial period value is 50%
                                                        ;   of full scale (for High)
0043 B01F                       MOVLW   0x1F
0044 4A1F                       MOVPF   W, PR1L         ; The initial period value is
                                                        ;   of full scale (for Low)
                        ;
                        ;
                        ; Default PWM values should be set, and the timer should be started
                        ; and the interrupts enabled.
                        ;
0045 B0F0                       MOVLW   0xF0            ; Load the Period register
0046 0114                       MOVWF   PR1             ;
0047 B803                       MOVLB   3               ; Select register Bank 3
0048 B0C0                       MOVLW   0xC0            ; Load the T1 duty cycle register
0049 0112                       MOVWF   PW1DCH          ;
004A 0110                       MOVWF   PW1DCL          ; effectively loaded with 0
```


© 1993 Microchip Technology Inc.

```
004B B031                       MOVLW   0x31              ;** Enable PWM1 and PWM2 outputs
004C 0117                       MOVWF   TCON2             ;**   and turn on Timer1.
004D 8307                       BSF     INTSTA,3          ; Turn on Peripheral Interrupts
004E B801                       MOVLB   1                 ; Select register Bank 1
004F B010                       MOVLW   0x10              ; Enable Timer1 overflow
0050 0117                       MOVWF   PIE               ;   Interrupts (when GLINTD = 0)
0051 8C06                       BCF     CPUSTA,4          ; Enable ALL interrupts
0052 B800                       MOVLB   0                 ; Bank 0

                        ;
                        ; Only need to update PWM values on the first occurance of a new level on
                        ; Else loop waiting for level to change.
                        ;
0053 8827               HIGH1ST BCF     FLAG_REG, 0       ; First time in loop (this cycle) = True
0054 9012               HIGHCYC BTFSS   PORTB,  0         ; Is PortB0 low
0055 C05F                       GOTO    LOW1ST            ; PORTB0 = L
0056 9827                       BTFSC   FLAG_REG, 0       ; Is this the First High time(this cycle)?
0057 C054                       GOTO    HIGHCYC           ; Loop looking for low signal on PortB0
0058 8027                       BSF     FLAG_REG, 0       ; Set First time in loop(this cycle)=False
                        ;
                        ; Here is where we update the PWM values (period and Duty cycle) for high
                        ;
0059 B803                       MOVLB   3                 ; Bank 3
005A 5A20                       MOVPF   DC1H, NEW_DC1     ;
005B 5B21                       MOVPF   DC1QH, NEW_DC1Q   ;
005C 5C22                       MOVPF   PR1H, NEW_PR1     ;
005D E06B                       CALL    PWM1_UD           ;
005E C054                       GOTO    HIGHCYC           ; Loop looking for low signal on
                        ;
                        ;
005F 8827               LOW1ST  BCF     FLAG_REG, 0       ; First time in loop (this
0060 9812               LOWCYC  BTFSC   PORTB, 0           ; Is PortB0 high
0061 C053                       GOTO    HIGH1ST           ; PORTB0 = H
0062 9827                       BTFSC   FLAG_REG, 0       ; Is this the First Low time
0063 C060                       GOTO    LOWCYC            ; Loop looking for high signal on PortB0
0064 8027                       BSF     FLAG_REG, 0       ; First time in loop (this    ;
                        ; Here is where we update the PWM values (period and Duty cycle) for low
                        ;
0065 B803                       MOVLB   3                 ; Bank 3
0066 5D20                       MOVPF   DC1L, NEW_DC1     ;
0067 5E21                       MOVPF   DC1QL, NEW_DC1Q   ;
0068 5F22                       MOVPF   PR1L, NEW_PR1     ;
0069 E06B                       CALL    PWM1_UD           ;
006A C060                       GOTO    LOWCYC            ; Loop looking for high signal

                        ;
                        ; This code segment ensure that all PWM values (period and duty cycle)
                        ; are updated at the same time. This is done by ensuring that the Timer
                        ; is at least PWM_WIN (0Dh) cycles before the PR1 value (PR1 - PWM_WIN >
                        ; If not a "glitch" could occur in the PWM waveform. When only the 1st duty
                        ; cycle register is latched for this PWM cycle, and the following PWM period
                        ; will latch the 2nd duty cycle register.
                        ;
006B 8406               PWM1_UD BSF     CPUSTA, 4         ; Disable Global Interrupts
006C B802                       MOVLB   2                 ; Bank 2
006D 6A10                       MOVFP   TMR1, W           ; Load W reg. with Timer1 value
006E 0414                       SUBWF   PR1, 0            ; PR1 - TMR1 -> W reg.
006F 3025                       CPFSLT  PWM_WIN           ; Check if Timer1 is about to
0070 C06B                       GOTO    PWM1_UD           ; Overflow would have occurred
                                                          ;   PWM updates, Delay a few
0071 B803                       MOVLB   3                 ; Bank 3
0072 6A20                       MOVFP   NEW_DC1, W        ; Your New PWM MSB
0073 0112                       MOVWF   PW1DCH            ; Loaded in duty cycle buffer
0074 6A21                       MOVFP   NEW_DC1Q, W       ; Your New PWM LSB
0075 0110                       MOVWF   PW1DCL            ; Loaded in duty cycle buffer
0076 B801                       MOVLB   1                 ; Back to Bank 1
0077 8C16                       BCF     PIR, 4            ; Clear T1 Overflow interrupt
```


© 1993 Microchip Technology Inc.

```
0078 8417                       BSF     PIE, 4          ; Enable T1 int
0079 8C06                       BCF     CPUSTA, 4       ; Enable Global Interrupts
007A B800                       MOVLB   0               ; Bank 0
007B 0002                       RETURN                  ;** this does not need to be
                                                        ;**  as a subroutine.
                        ;
                        ; Other Interrupt routines. (Not utilized in this example)
                        ;
007C 0005               EXT_INT         RETFIE          ; RA0/INT interrupt routine
                                                        ;   (NOT used in this program)
007D 0005               RTCCINT         RETFIE          ; RTCC overflow interrupt routine
                                                        ;   (NOT used in this program)
007E 0005               RT_INT          RETFIE          ; RA1/RT interrupt routine
                                                        ;   (NOT used in this program)
                                                        ;
007F C02B               SRESET          GOTO    START   ; If program became lost, goto
                                                        ;   START and reinitalize.
                        ;
                        ;
                        ; When the executed address is NOT in the program range, the
                        ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                        ; this location you could branch to a routine to recover or
                        ; shut down from the invalid program execution.
                        ;
                                        ORG     END_OF_PROG_MEM ;
07FF C07F                               GOTO    SRESET          ; The program has lost it's mind,
                                                                ; do a system reset
                                        END


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

MICROCHIP

AN545

Using the Capture Module

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application report provides examples which illustrate the uses of input capture using the PIC17C42 Timer3 module. These examples may be modified to suit the specific needs of an application.

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

TIMER3 DESCRIPTION

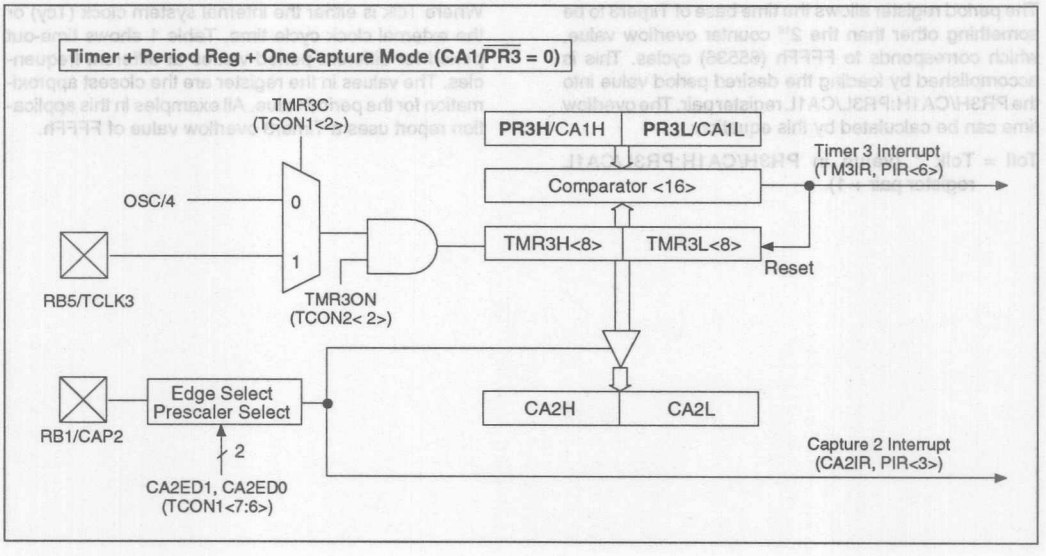
Timer3 is a 16-bit counter that has two modes of operation which can be software selected. The CA1/$\overline{PR3}$ bit (TCON2<3>) selects the mode of operation. The two modes are:

1. Timer3 with Period Register and Single Capture Register (see Figure 1)
2. Timer3 and Dual Capture Registers (see Figure 2)

Timer3 is the time-base for capture operations.

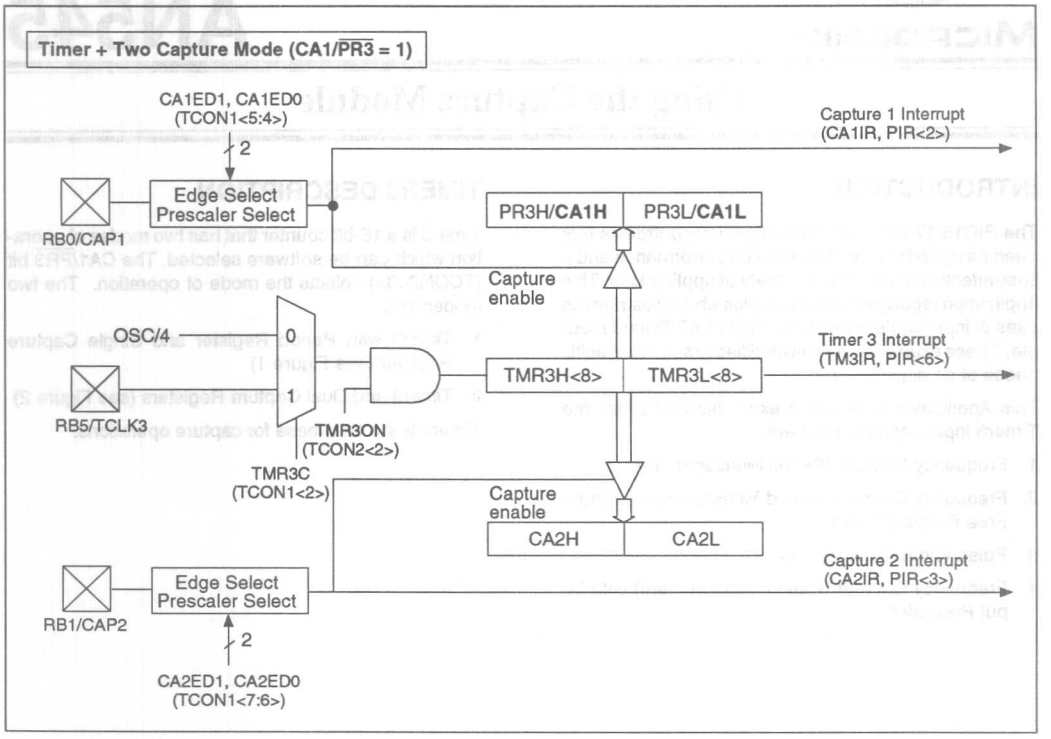
4

FIGURE 1 - TIMER3 WITH PERIOD REGISTER AND SINGLE CAPTURE REGISTER



© 1993 Microchip Technology Inc.

FIGURE 2 - TIMER3 AND DUAL CAPTURE REGISTERS



The period register allows the time base of Timer3 to be something other than the 2^{16} counter overflow value, which corresponds to FFFFh (65536) cycles. This is accomplished by loading the desired period value into the PR3H/CA1H:PR3L/CA1L register pair. The overflow time can be calculated by this equation:

TofI = Tclk * (value in PR3H/CA1H:PR3L/CA1L register pair + 1).

Where Tclk is either the internal system clock (Tcy) or the external clock cycle time. Table 1 shows time-out period for different period values at different frequencies. The values in the register are the closest approximation for the period value. All examples in this application report uses a Timer3 overflow value of FFFFh.

© 1993 Microchip Technology Inc.

TABLE 1 - TIMER3 OVERFLOW TIMES

	Period Register					
Overflow Time	**@ 16 MHz (250 ns)**	**@ 10 MHz (400 ns)**	**@ 8 MHz (500 ns)**	**@ 5 MHz (800 µs)**	**@ 2 MHz (2.0 µs)**	**@ 32 KHz (125 µs)**
8.192 S	N.A.	N.A.	N.A.	N.A.	N.A.	0xFFFF
131.072 mS	N.A.	N.A.	N.A.	N.A.	0xFFFF	0x0418
52.428 mS	N.A.	N.A.	N.A.	0xFFFF	0x6666	0x01A3
32.7675 mS	N.A.	N.A.	0xFFFF	0x9FFF	0x3FFF	0x0106
26.214 mS	N.A.	0xFFFF	0xCE20	0x80D4	0x3388	0x00D3
16.384 mS	0xFFFF	0xA000	0x8000	0x5000	0x2000	0x0083
10.0 mS	0x9C40	0x61A8	0x4E20	0x30D4	0x1388	0x0050
4.0 mS	0x3E80	0x2710	0x1F40	0x1388	0x07D0	0x0020
1.0 mS	0x0FA0	0x09C4	0x07D0	0x04E2	0x01F4	0x0008
600 uS	0x0960	0x05DC	0x04B0	0x02EE	0x012C	0x0005
100 uS	0x0190	0x00FA	0x00C8	0x007D	0x0032	N.A.

The uses of an input capture are all for time based measurements. These include:

- Frequency measurement
- Duty cycle and pulse width measurements

The PIC17C42 has two pins (RB0/CAP1 and RB1/CAP2) which can be used for capturing the Timer3 value, when a specified edge occurs. The input capture can be specified to occur on one of the following four events:

- Falling Edge
- Rising Edge
- 4th Rising Edge
- 16th Rising Edge

These are specified, for both capture pins, by the register TCON1<7:4>.

This flexibility allows an interface without the need of additional hardware to change polarity or specify an input prescaler.

4

© 1993 Microchip Technology Inc.

The control registers that are utilized for by Timer3 are shown in Table 2. Shaded Boxes are control bits that are not used by the Timer3 module, the Peripheral Interrupt enable and flag bits, and the Global Interrupt enable bit.

TABLE 2 - REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE:

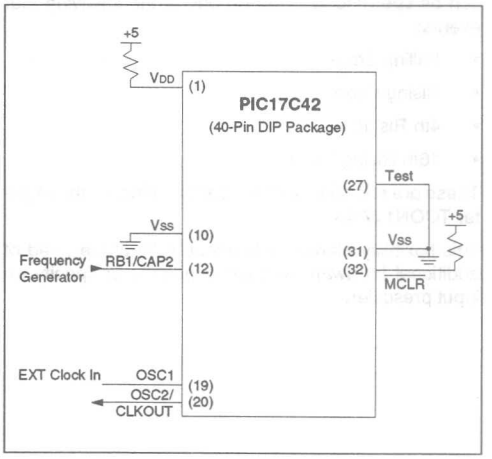
Name	BANK	ADDR	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
INTSTA	All	6h	PEIR	RTXIR	RTCIR	INTIR	PEIE	RTXIE	RTCIE	INTIE
CPUSTA	All	7h	-	-	STKAV	GLINTD	TO	PD	-	-
TMR3L	2	12h	Timer3 LSB Register							
TMR3H	2	13h	Timer3 MSB Register							
CA2L	3	14h	Timer3 Capture2 LSB Register							
CA2H	3	15h	Timer3 Capture2 MSB Register							
PIR	1	16h	IRB	TM3IR	TM2IR	TM1IR	CA2IR	CA1IR	TBMT	RBFL
PIE	1	17h	IEB	TM3IE	TM2IE	TM1IE	CA2IE	CA1IE	TXIE	RCIE
PR3L/ CA1L	2	16h	Timer3 - Period MSB Register/Capture1 MSB Register							
PR3H/ CA1H	2	17h	Timer3 - Period MSB Register/ Capture1 MSB Register							
TCON1	3	16h	CA2ED1	CA2ED0	CA1ED1	CA1ED0	16/8	TMR3C	TMR2C	TMR1C
TCON2	3	17h	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/PR3	TMR3ON	TMR2ON	TMR1ON

This Application Note has 4 examples that utilize the Timer3 input capture. They are:

1. Frequency Counter (Period Measurement)
2. Frequency Counter (Period Measurement) using a Free Running Timer
3. Pulse Width Measurement
4. Frequency Counter (Period Measurement) with Input Prescaler

All these examples can be run from a simple setup, this is show in Figure 3.

FIGURE 3 - APPLICATION HARDWARE SETUP



A discussion of each application with the operation of the software and application issues. The source listings for these are in appendices A-D.

© 1993 Microchip Technology Inc.

PERIOD MEASUREMENT (FREQUENCY COUNTER)

Period measurement is simply done by setting the counter to 0000h, then starting the counter on the 1st rising edge. On the following rising edge, the capture 2 register is loaded with the Timer3 value and the Timer3 (TMR3) register is cleared. If the period is greater than the overflow rate of Timer3, the timer overflows, causing an interrupt. With a TMR3 overflow, an interrupt occurs and the overflow counter may need to be incremented. The overflow counter should be incremented if:

1. The TMR3 overflow is the only interrupt source.
2. Both the TMR3 overflow and capture2 interrupts occurred at near the same time, but the TMR3 overflow occurred first (Most Significant Byte of the Capture2 register = 00h).

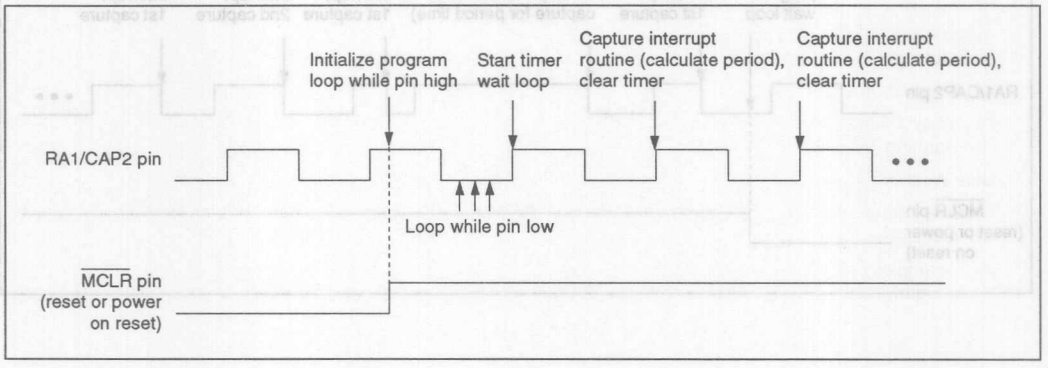
Once a capture2 has occurred, the capture registers are moved to data RAM, the capture2 interrupt flag is cleared and the TMR3 register is loaded with an offset value. This offset value is the number of cycles from the time the interrupt routine is entered to when the TMR3 register is reloaded. In this example a data RAM location is used as an overflow counter. This gives in effect a 24-bit timer. The software flow for this routine is shown in Figure 4.

The program listing in Appendix A implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	30 Words
RAM used:	4 Bytes
Maximum frequency that can be measured:	130 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (±250 nsec)

4

FIGURE 4 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2



© 1993 Microchip Technology Inc.

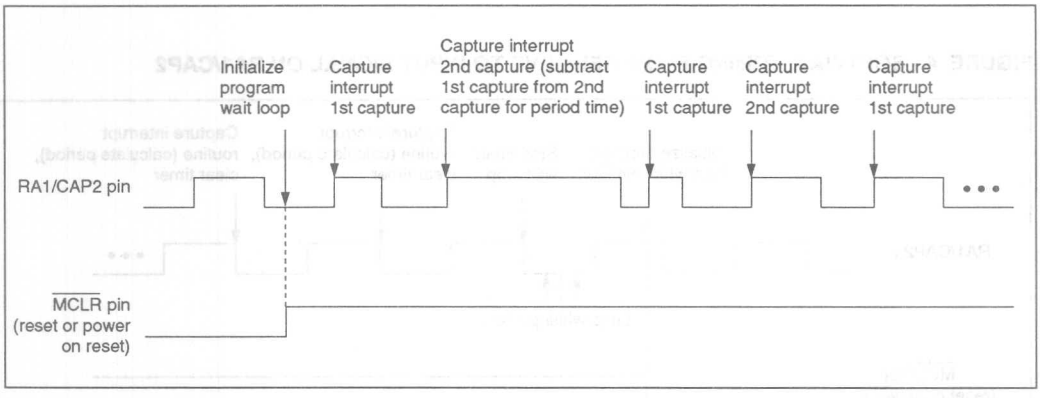
PERIOD MEASUREMENT (FREQUENCY COUNTER) USING A FREE RUNNING TIMER

In many applications the timer (TMR3) would need to be used for multiple tasks, and could not be reset (modified) by any one of these tasks. This is called a free running timer. To do period measurement in a free running timer application, the program needs to store each capture in a data RAM location pair (word). The 1st capture in data RAM locations input capture2A (IC2AH:IC2AL) and the 2nd capture in data RAM locations input capture2B (IC2BH:IC2BL). Once the two captures have occurred, the values in these two words is subtracted. Since this is a free running timer, the value in input capture2B may be less than the value in input capture2A. This is if the 1st capture occurs, then the Timer3 overflows, and then the 2nd capture occurs. So an overflow counter should only be incremented if the Timer3 overflow occurs after a capture1 but before the capture2 occurs. With the use of an overflow counter this becomes an effective 24-bit period counter. The software flow for this routine is shown in Figure 5.

The program listing in Appendix B implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	80 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (±250 nsec)

FIGURE 5 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2



© 1993 Microchip Technology Inc.

PULSE WIDTH MEASUREMENT USING A FREE RUNNING TIMER

Applications that require the measurement of a pulse width can also be easily handled. The PIC17C42 can be programmed to measure either the low or the high pulse. The software example in Appendix C measures the High pulse time. The program is initialized to capture on the rising edge of the RA1/CAP2 pin. After this event occurs, the capture mode is switched to the falling edge of the RA1/CAP2 pin. When the capture edge is modified (rising to falling, or falling to rising) a capture interrupt is generated. This "false" interrupt request must be cleared before leaving the interrupt service routine, or the program will immediately re-enter the interrupt service routine due to this "false" request. When the falling edge of the RA1/CAP2 pin occurs, the difference of the two capture values is calculated. The flow for this is shown in Figure 6.

Due to the software overhead of the Peripheral Interrupt routine the following are the limitations on the input signal on RA1/CAP2 pin. This does not include any software overhead that may be required in the main routine, or if additional peripheral interrupt features need to be included. This is shown in Table 3. If you assume that the input is a square wave (high time = low time), one needs to take the worst case time of the two minimum pulse times (11 uS) times two, to determine the period. The maximum continuous input frequency would then be approximately 45.5 KHz. For a single pulse measurement, minimum pulse width is 4.5 µs.

The program listing in Appendix C implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	51 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	71 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (±250 nsec)

4

FIGURE 6 - SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RA1/CAP2

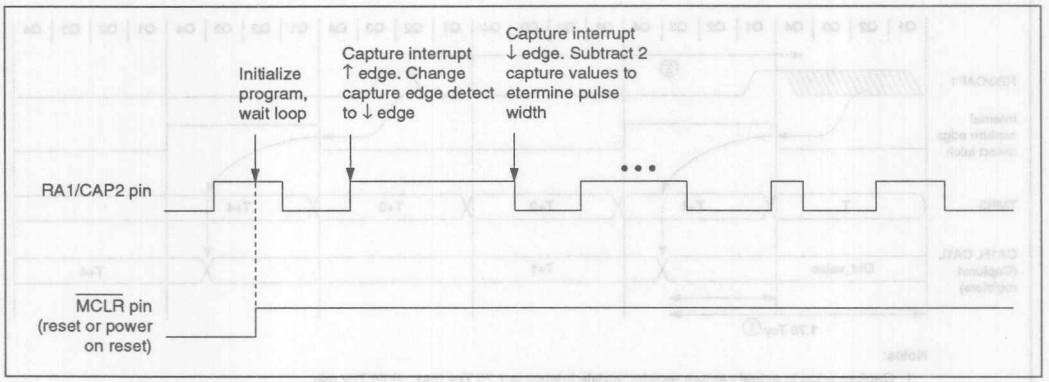


TABLE 3 - PERIPHERAL INTERRUPT ROUTINE

EVENT		# of Cycles	Time @ 16 MHz
1st CAPTURE	Capture1 Only	18	4.5 uS
	Capture1 and Timer Overflow	30	7.5 uS
2nd CAPTURE	Capture Only	35	8.75 uS
	Capture and Timer Overflow	41	10.25 uS
Minimum Pulse High	Capture1 and Timer Overflow + INT Latency	33	8.25 uS
Minimum Pulse Low	Capture2 and Timer Overflow + INT Latency	44	11 uS
Minimum Period (square wave)	2 * (Minimum Pulse Low)	88	22 uS

© 1993 Microchip Technology Inc.

PERIOD MEASUREMENT (FREE RUNNING TIMER) WITH A PRESCALER

Occasionally the application may require a prescaler on the input signal. This may be due to application requirements, such as:

- Require higher resolution measurement of the input signal
- Reduce interrupt service overhead
- The input frequency is higher than interrupt service routine

The software selectable prescaler of the PIC17C42 allows the designer to easily implement this in their system without the cost of additional hardware. Care must be taken in determining if this option is appropriate. For example, if the input frequency is not stable (excessive frequency change per period) then the prescaler will give a less accurate capture value than the individual measurements.

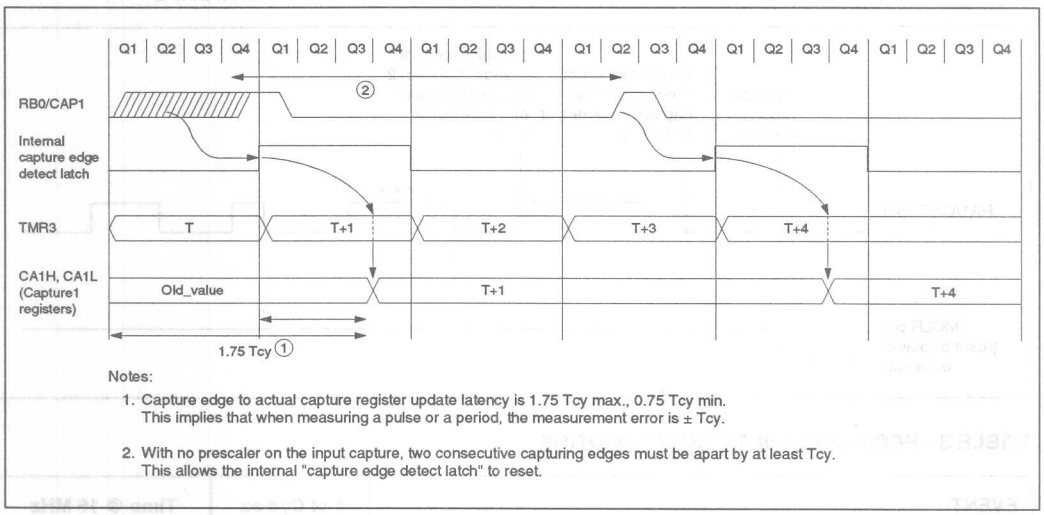
In cases where the resolution of the input frequency is important, the Prescaler can be used to reduce the input capture error. There are two components to input capture:

1. Resolution Error
2. Input Synchronization Error

These two errors add together to form the total capture error. Resolution error is dependent on the rate at which the timer is incremented, remember the timer may be based on an external clock (must be slower than Tcy). The input synchronization error is dependent on the system clock speed (Tcy), and will be less than Tcy.

It is easy to see that when a capture occurs the synchronization error (Tesync) can be up to 1 Tcy (see Figure 7). This error is constant regardless of the number of edges that occur before the capture is taken. So a capture on the 1st edge gives a synchronization error per sample up to Tcy. While a capture taken on the 16th edge gives a synchronization error per sample only up to Tcy / 16, by achieving a smaller percentage of error, the captured value becomes more accurate.

FIGURE 7 - SYNCHRONIZATION ERROR WITH NO CAPTURE PRESCALER



Notes:

1. Capture edge to actual capture register update latency is 1.75 Tcy max., 0.75 Tcy min. This implies that when measuring a pulse or a period, the measurement error is ± Tcy.
2. With no prescaler on the input capture, two consecutive capturing edges must be apart by at least Tcy. This allows the internal "capture edge detect latch" to reset.

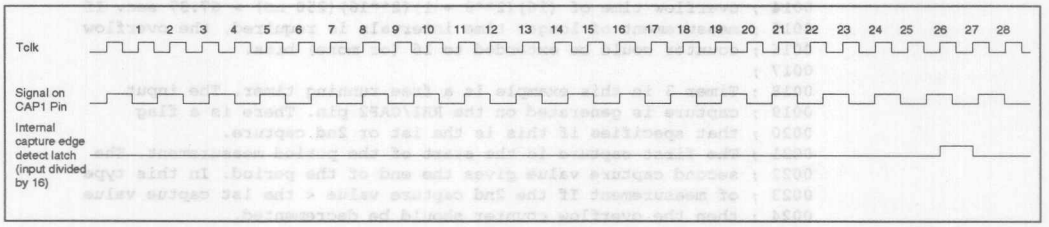
© 1993 Microchip Technology Inc.

Another scenario is when the signal on the input capture pin is different than the system cycle time (Tcy), for example 1.6 Tcy. If you tried to capture this you would have a capture value of 1. If you set the prescaler to actually capture on the 16th edge you would have 16*1.6 Tcy = 25.6 Tcy, which would be latched on the 26th Tcy (see Figure 8). This 0.4 Tcy error is over 16 samples, which therefore gives an effective error/sample of 0.025 Tcy.

The program listing in Appendix D implements this, assuming only Timer3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	80 KHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	+/- Tcy (± 16.625 nsec)

FIGURE 8 - INPUT CAPTURE DIVIDED BY 16 PRESCALE EXAMPLE



4

Author: Mark Palmer
Logic Products Division

© 1993 Microchip Technology Inc.

APPENDIX A - PERIOD MEASUREMENT EXAMPLE CODE

```
MPASM 01.01 Released  IC_D16_2.ASM   6-7-1994  11:17:12                    PAGE  1

      LOC  OBJECT CODE     LINE SOURCE TEXT
                  0001          LIST    p=17C42, c=132
                  0002 ; This is the basic outline for a program that can determine the
                  0003 ; frequency of an input, via input capture. The input capture has
                  0004 ; been selected to capture on the 16th rising edge. This is useful
                  0005 ; for high frequency inputs, where an interrupt on each rising edge
                  0006 ; would not be able to be serviced (at that rate). This particular
                  0007 ; example can support an input signal with a period of approximatly
                  0008 ; 625 nS. Without the divide by 16 selected, this is approximatly
                  0009 ; 10 us. This period time increases (frequency decreases) as the
                  0010 ; overhead in the main routine increases.
                  0011 ;
                  0012 ; This routine uses an 8-bit register to count the times that timer3
                  0013 ; overflowed. At the Max crystal frequency of 16 MHz, this gives an
                  0014 ; overflow time of (16)(2**8 + 1)(2**16)(250 nS) > 67.37 sec. If
                  0015 ; measurement of longer time intervals is required, the overflow
                  0016 ; counter could be extended to 16 (or more) bits.
                  0017 ;
                  0018 ; Timer 3 in this example is a free running timer. The input
                  0019 ; capture is generated on the RB1/CAP2 pin. There is a flag
                  0020 ; that specifies if this is the 1st or 2nd capture.
                  0021 ; The first capture is the start of the period measurement. The
                  0022 ; second capture value gives the end of the period. In this type
                  0023 ; of measurement If the 2nd capture value < the 1st captue value
                  0024 ; then the overflow counter should be decremented.
                  0025 ;
                  0026 ;  Program:   IC_D16_2.ASM
                  0027 ;  Revision:  1.0
                  0028 ;
                  0029 ; Do the EQUate table
                  0030 ;
0020              0031 IC2OF           EQU     0x20            ; T3 overflow register
0021              0032 IC2BH           EQU     0x21            ; T3 ICA2 MSB register (2nd Cap)
0022              0033 IC2BL           EQU     0x22            ; T3 ICA2 LSB register
0023              0034 IC2AH           EQU     0x23            ; T3 ICB2 MSB register (1st Cap)
0024              0035 IC2AL           EQU     0x24            ; T3 ICB2 LSB register
0025              0036 T3OFLCNTR       EQU 0x25                ; Temperay T3 overflow register
0026              0038 FLAG_REG        EQU 0x26                ; Register that has the Flag bits
                  0039 ;
                  0040 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                  0041 ;                    -   -   -   -   -   -  UFL CAP1
                  0042 ;    CAP1 = 0,  1st Capture
                  0043 ;         = 1,  2nd Capture
                  0044 ;
                  0045 ;    UFL = 0,  No Underflow
                  0046 ;        = 1,  Underflow during subtract
                  0047 ;
07FF              0048 END_OF_PROG_MEM         EQU 0x07FF
                  0049 ;
                  0050 ;
0004              0051 ALUSTA          EQU     0x04
0006              0052 CPUSTA          EQU     0x06
0007              0053 INTSTA          EQU     0x07
000A              0054 W               EQU     0x0A
                  0055 ;
0012              0056 PORTB           EQU     0x12            ; Bank 0
                  0057 ;
0016              0058 PIR             EQU     0x16            ; Bank 1
0017              0059 PIE             EQU     0x17
                  0060 ;
0012              0061 TMR3L           EQU     0x12            ; Bank 2
0013              0062 TMR3H           EQU     0x13
0016              0063 T3PRL           EQU 0x16
0017              0064 T3PRH           EQU 0x17
```


© 1993 Microchip Technology Inc.

```
                        0065 ;
0014                    0066 CA2L           EQU     0x14            ; Bank 3
0015                    0067 CA2H           EQU     0x15
0016                    0068 TCON1          EQU     0x16
0017                    0069 TCON2          EQU     0x17
                        0071                ORG     0x0000          ; Origin for the RESET vector
0000 C028               0072                GOTO    START           ; On reset, go to the start of
                        0073                                        ;   the program
                        0074                ORG     0x0008          ; Origin for the external RA0/INT
                        0075                                        ;   interrupt vector
0008 C068               0076                GOTO    EXT_INT         ; Goto the ext. interrupt
                        0077                                        ;   on RA0/INT routine
                        0078                ORG     0x0010          ; Origin for the RTCC
                        0079                                        ;   overflow interrupt vector
0010 C069               0080                GOTO    RTCCINT         ; Goto the RTCC overflow interrupt
                        0081                                        ;   routine
                        0082                ORG     0x0018          ; Origin for the external
                        0083                                        ;   RA1/RT interrupt vector
0018 C06A               0084                GOTO    RT_INT          ; Goto the ext. interrupt on
                        0085                                        ;   RA1/RT routine
                        0086                ORG     0x0020          ; Origin for the interrupt vector
                        0087                                        ;   of any enabled peripheral
0020 C03E               0088                GOTO    PER_INT         ; Goto the interrupt from a
                        0089                                        ;   peripheral routine
                        0091                ORG     0x0028          ; Origin for the top of
                        0092                                        ;   program memory
0028 8406               0093 START          BSF     CPUSTA,4        ; Disable ALL interrupts via the
                        0094                                        ;   Global Interrupt Disable
                        0095                                        ;   (GLINTD) bit.
                        0096                                        ;
                        0097 MAIN                                   ; Place Main program here
0029 B803               0098                MOVLB   3               ; Select register Bank 3
002A 2817               0099                CLRF    TCON2,0         ; Stop the timers, Single Capture
002B B0F0               0100                MOVLW   0x0F0           ; Initalize TCON1 so that
002C 0116               0101                MOVWF   TCON1           ;   T1 (8-bit), T2 (8-bit),
                        0102                                        ;   and T3 run off the internal
                        0103                                        ;   system clock. Capture2
                        0104                                        ;   captures on the 16th rising edg
                        0105 ;
                        0106 ; Initalize Timer 3, load the timer with the number of cycles that
                        0107 ; the device executes (from RESET) before the timer is turned on
                        0108 ; Therefore the offset is required due to software overhead.
                        0109 ;
002D B802               0110                MOVLB   2               ; Select register Bank 2
002E 280A               0111                CLRF    W,0             ; Clear the W register
002F 0126               0112                MOVWF   FLAG_REG        ; Initalize to 0
0030 0113               0113                MOVWF   TMR3H           ; Timer3 MSB = 0
0031 B000               0114                MOVLW   0x00            ; Timer3 LSB = Offset
0032 0112               0115                MOVWF   TMR3L           ;
                        0116 ;
                        0117 ; Load the Timer 3 period register with 0xFFFF, which will give an
                        0118 ; interrupt on the overflow of Timer3
                        0119 ;
0033 B0FF               0120                MOVLW   0xFF            ;
0034 0117               0121                MOVWF   T3PRH           ;
0035 0116               0122                MOVWF   T3PRL           ;
                        0123 ;
                        0124 ; the timer should be started and interrupts enabled.
                        0125 ;
0036 B803               0126                MOVLB   3               ; Select register Bank 3
0037 8217               0127                BSF     TCON2,2         ; Turn on timer 3.
0038 8307               0128                BSF     INTSTA,3        ; Turn on Peripheral Interrupts
0039 B801               0129                MOVLB   1               ; Select register Bank 1
003A B048               0130                MOVLW   0x48            ; Enable Caputure 2 and Timer3
003B 0117               0131                MOVWF   PIE             ;    Interrupts (when GLINTD = 0)
                        0132 ;
                        0133 ; This is where you would do the things you wanted to do.
                        0134 ; this example will only loop waiting for the interrupts.
                        0135 ;
```


© 1993 Microchip Technology Inc.

```
003C 8C06          0136 WAIT          BCF     CPUSTA,4        ; Enable ALL interrupts
003D C03C          0137               GOTO    WAIT            ; Loop here waiting for a timer
                   0138                                       ;   Interrupt
                   0140 ;
                   0141 ; The interrupt routine for any peripheral interrupt, This routine
                   0142 ; only deals with Timer3 (T3) interrupts.
                   0143 ;
                   0144 ; Time required to execute interrupt routine. Not including
                   0145 ; interrupt latency (time to enter into the interrupt routine)
                   0146 ;
                   0147 ;   case1 - only T3 overflow              = 12 cycles
                   0148 ;   case2 - 1st capture                   = 14 cycles
                   0149 ;   case3 - 2nd capture                   = 30 cycles
                   0150 ;   case4 - T3 overflow and 1st capture   = 34 cycles
                   0151 ;   case5 - T3 overflow and 2nd capture   = 50 cycles
                   0152 ;
                   0153 ;
003E B801          0154 PER_INT MOVLB   1                         ; Select register Bank 1
003F 9E16          0155               BTFSC   PIR,6               ; Did T3 overflow?
                   0156                                       ;   If not skip next Instruction
0040 C055          0157               GOTO    T3OVFL          ; Inc overflow cntr and clear flag
0041 9316          0158 CK_CAP        BTFSS   PIR,3           ; Did the RB1/CAP2 pin cause an
                   0159                                       ;   interrupt?
0042 0005          0160               RETFIE                  ; No RB1/CAP2 interrupt,
                   0161                                       ;   Return from Interrupt
                   0162 ;
                   0163 ; This potion of the code takes the 1st capture and stores its
                   0164 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                   0165 ; is take, its value is stored in register pair IC2BH:IC2BL.
                   0166 ; A 16-bit subtract is performed, with the final 24-bit result
                   0167 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                   0168 ; be correct after the next capture occurs (IC2BH:IC2BL will
                   0169 ; change), so the main routine must utilize this value before
                   0170 ; it changes.
                   0171 ;
0043 8B16          0172 CAPTURE BCF             PIR,3           ; Clear Capture2 interrupt flag
0044 B803          0173               MOVLB   3               ; Select register Bank 3
0045 9826          0174               BTFSC   FLAG_REG,0      ; 1st or 2nd capture2?
0046 C04B          0175               GOTO    CAP2            ; It was the 2nd Capture
0047 5424          0176 CAP1    MOVPF   CA2L,IC2AL              ; Move the captured value to
0048 5523          0177               MOVPF   CA2H,IC2AH      ;   temporary registers
0049 8026          0178               BSF     FLAG_REG,0      ; Have 1st capture2
004A 0005          0179               RETFIE                  ; Return from Interrupt
                   0180                                       ;
004B 5422          0182 CAP2    MOVPF   CA2L,IC2BL              ; Move the captured value to
004C 5521          0183               MOVPF   CA2H,IC2BH      ;   temporary registers
                   0184                                       ;   (to pervent being overwritten)
                   0185                                       ;
004D E061          0186               CALL    SUB16           ; Call routine which subtracts
                   0187                                       ;   2 16-bit numbers.
004E 9926          0188               BTFSC   FLAG_REG,1      ; Underflow during SUB16?
004F 0725          0189               DECF    T3OFLCNTR,1     ; Since underflow, decrement the
                   0190                                       ;   overflow counter.
0050 2926          0191               CLRF    FLAG_REG,1      ; Clear the flag bits for
                   0192                                       ;   underflow and 2nd capture2
0051 6A25          0193               MOVFP   T3OFLCNTR,W     ; Store the T3 input capture
0052 4A20          0194               MOVPF   W,IC2OF         ;   overflow value in IC2OF
0053 2825          0195               CLRF    T3OFLCNTR,0     ; Clear the Data register which
                   0196                                       ;   counts how many times Timer 3
                   0197                                       ;   overflows.
0054 0005          0198               RETFIE                  ; Return from interrupt
                   0199 ;
                   0200 ; When Timer 3 has overflowed, the overflow counter only should
                   0201 ; be incremented when the overflow occurs after a capture 1
                   0202 ; but before the capture 2. The 4 possible cases when entering
                   0203 ; the T3OVFL section of the PER_INT routine are as follows:
                   0204 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                   0205 ;           for Capture 1 to occur). Do Not increment counter
                   0206 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
```


© 1993 Microchip Technology Inc.

```
                    0207 ;                for Capture 2 to occur). Increment counter
                    0208 ;   Case 3: T3 Overflow happened after Capture. Do Not
                    0209 ;            increment overflow counter
                    0210 ;   Case 4: T3 Overflow occured before Capture 2 and FLAG_REG.0 = 1
                    0211 ;            (waiting for Capture 2 to occur). Increment counter
                    0212
                    0213 ;
0055 8E16           0214 T3OVFL          BCF     PIR,6          ; Clear Overflow interrupt flag
0056 9316           0215                 BTFSS   PIR,3          ; Did the RB1/CAP2 pin also
                    0216                                        ;   cause an interrupt?
0057 C05E           0217                 GOTO    FR0            ; No, Check if between 1st
                    0218                                        ;   and 2nd capture
0058 B803           0219                 MOVLB   3              ; Bank 3
0059 280A           0220                 CLRF    W,0            ; W = 0
005A 3115           0221                 CPFSEQ  CA2H           ; if CA2H = 0, overflow happened
005B C05E           0222                 GOTO    FR0            ;   first, must check FLAG_REG
                    0223                                        ;   bit 0
005C B801           0224                 MOVLB   1              ; Back to bank 1
005D C043           0225                 GOTO    CAPTURE        ; Capture happened first, do NOT
                    0226                                        ;   Increment overflow counter
                    0227                                        ;   and do capture routine
005E 9826           0228 FR0             BTFSC   FLAG_REG,0     ; Between Capture 1 and Capture 2?
005F 1525           0229                 INCF    T3OFLCNTR,1    ; Yes, Inc. the overflow counter
0060 0005           0230                 RETFIE                 ; Return from overflow interrupt
                    0231 ;
0061 6A24           0232 SUB16   MOVFP   IC2AL,W                ; Do the 16-bit subtraction
0062 0522           0233                 SUBWF   IC2BL,1        ;
0063 6A23           0234                 MOVFP   IC2AH,W        ;
0064 0321           0235                 SUBWFB  IC2BH,1        ;
0065 9004           0236                 BTFSS   ALUSTA,0       ; Is the result pos. or neg. ?
0066 8126           0237                 BSF     FLAG_REG,1     ; neg., Set the underflow flag
0067 0002           0238                 RETURN                 ; Return from the subroutine
                    0240 ;
                    0241 ; Other Interrupt routines. (Not utilized in this example)
                    0242 ;
0068 0005           0243 EXT_INT         RETFIE                 ; RA0/INT interrupt routine
                    0244                                        ;   (NOT used in this program)
0069 0005           0245 RTCCINT         RETFIE                 ; RTCC overflow interrupt routine
                    0246                                        ;   (NOT used in this program)
006A 0005           0247 RT_INT          RETFIE                 ; RA1/RT interrupt routine
                    0248                                        ;   (NOT used in this program)
                    0249                                        ;
006B C028           0250 SRESET          GOTO    START          ; If program became lost, goto
                    0251                                        ;   START and reinitalize.
                    0252 ;
                    0253 ;
                    0254 ; When the executed address is NOT in the program range, the
                    0255 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                    0256 ; this location you could branch to a routine to recover or
                    0257 ; shut down from the invalid program execution.
                    0258 ;
                    0259 ORG      END_OF_PROG_MEM ;
07FF C06B           0260                 GOTO    SRESET         ; The program has lost it's mind,
                    0261                                        ;   do a system reset
                    0262                 END
                    0263
```

APPENDIX B - PERIOD MEASUREMENT WITH FREE RUNNING TIMER EXAMPLE CODE

```
MPASM 01.01 Released   IC_FRT2.ASM   6-7-1994  11:16:54                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    p=17C42, c=132
                     0002 ; This is the basic outline for a program that can determine the
                     0003 ; frequency of an input, via input capture. This routine uses an
                     0004 ; 8-bit register to count the times that timer3 overflowed. At the
                     0005 ; Max crystal frequency of 16 MHz, this gives an overflow time of
                     0006 ; (2**16)(256 + 1)(250 nS) > 4.21 sec or a frequncy < 0.25 Hz. If
                     0007 ; measurement of longer time intervals is required, the overflow
                     0008 ; counter could be extended to 16 (or more) bits.
                     0009 ;
                     0010 ; Timer 3 in this example is a free running timer. The input
                     0011 ; capture is generated on the RB1/CAP2 pin. There is a flag
                     0012 ; that specifies if this is the 1st or 2nd capture.
                     0013 ; The first capture is the start of the period measurement. The
                     0014 ; second capture value gives the end of the period. In this type
                     0015 ; of measurement If the 2nd capture value < the 1st captue value
                     0016 ; then the overflow counter should be decremented.
                     0017 ;
                     0018 ;  Program:    IC_FRT2.ASM
                     0019 ;  Revision:   1.0
                     0020 ;
                     0021 ; Do the EQUate table
                     0022 ;
0020                 0023 IC2OF           EQU     0x20            ; T3 overflow register
0021                 0024 IC2BH           EQU     0x21            ; T3 ICA2 MSB register (2nd Cap)
0022                 0025 IC2BL           EQU     0x22            ; T3 ICA2 LSB register
0023                 0026 IC2AH           EQU     0x23            ; T3 ICB2 MSB register (1st Cap)
0024                 0027 IC2AL           EQU     0x24            ; T3 ICB2 LSB register
0025                 0028 T3OFLCNTR       EQU     0x25            ; Temperay T3 overflow register
                     0029 ;
0026                 0030 FLAG_REG        EQU     0x26            ; Register that has the Flag bits
                     0031 ;
                     0032 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                     0033 ;                    -   -   -   -   -   -  UFL CAP1
                     0034 ;    CAP1 = 0,  1st Capture
                     0035 ;         = 1,  2nd Capture
                     0036 ;
                     0037 ;    UFL = 0,  No Underflow
                     0038 ;        = 1,  Underflow during subtract
                     0039 ;
07FF                 0040 END_OF_PROG_MEM EQU     0x07FF
                     0041 ;
                     0042 ;
0004                 0043 ALUSTA          EQU     0x04
0006                 0044 CPUSTA          EQU     0x06
0007                 0045 INTSTA          EQU     0x07
000A                 0046 W               EQU     0x0A
                     0047 ;
0012                 0048 PORTB           EQU     0x12            ; Bank 0
                     0049 ;
0016                 0050 PIR             EQU     0x16            ; Bank 1
0017                 0051 PIE             EQU     0x17
                     0052 ;
0012                 0053 TMR3L           EQU     0x12            ; Bank 2

0013                 0054 TMR3H           EQU     0x13
0016                 0055 T3PRL           EQU 0x16
0017                 0056 T3PRH           EQU 0x17
                     0057 ;
0014                 0058 CA2L            EQU     0x14            ; Bank 3
0015                 0059 CA2H            EQU     0x15
0016                 0060 TCON1           EQU     0x16
0017                 0061 TCON2           EQU     0x17
```


© 1993 Microchip Technology Inc.

```
                0063            ORG     0x0000          ; Origin for the RESET vector
0000 C028       0064            GOTO    START           ; On reset, go to the start of
                0065                                    ;   the program
                0066            ORG     0x0008          ; Origin for the external RA0/INT
                0067                                    ;   interrupt vector
0008 C068       0068            GOTO    EXT_INT         ; Goto the ext. interrupt
                0069                                    ;   on RA0/INT routine
                0070            ORG     0x0010          ; Origin for the RTCC
                0071                                    ;   overflow interrupt vector
0010 C069       0072            GOTO    RTCCINT         ; Goto the RTCC overflow interrupt
                0073                                    ;   routine
                0074            ORG     0x0018          ; Origin for the external
                0075                                    ;   RA1/RT interrupt vector
0018 C06A       0076            GOTO    RT_INT          ; Goto the ext. interrupt on
                0077                                    ;   RA1/RT routine
                0078            ORG     0x0020          ; Origin for the interrupt vector
                0079                                    ;   of any enabled peripheral
0020 C03E       0080            GOTO    PER_INT         ; Goto the interrupt from a
                0081                                    ;   peripheral routine
                0083            ORG     0x0028          ; Origin for the top of
                0084                                    ;   program memory
0028 8406       0085 START      BSF     CPUSTA,4        ; Disable ALL interrupts via the
                0086                                    ;   Global Interrupt Disable
                0087                                    ;   (GLINTD) bit.
                0088                                    ;
                0089 MAIN                               ; Place Main program here
0029 B803       0090            MOVLB   3               ; Select register Bank 3
002A 2817       0091            CLRF    TCON2,0         ; Stop the timers, Single Capture
002B B070       0092            MOVLW   0x070           ; Initalize TCON1 so that
002C 0116       0093            MOVWF   TCON1           ;   T1 (8-bit), T2 (8-bit),
                0094                                    ;   and T3 run off the internal
                0095                                    ;   system clock. Capture2
                0096                                    ;   captures on the rising edge.
                0097 ;
                0098 ; Initalize Timer 3, load the timer with the number of cycles that
                0099 ; the device executes (from RESET) before the timer is turned on
                0100 ; Therefore the offset is required due to software overhead.
                0101 ;
002D B802       0102            MOVLB   2               ; Select register Bank 2
002E 280A       0103            CLRF    W,0             ; Clear the W register
002F 0126       0104            MOVWF   FLAG_REG        ; Initalize to 0
0030 0113       0105            MOVWF   TMR3H           ; Timer3 MSB = 0
0031 B013       0106            MOVLW   0x13            ; Timer3 LSB = Offset
0032 0112       0107            MOVWF   TMR3L           ;
                0108 ;
                0109 ; Load the Timer 3 period register with 0xFFFF, which will give an
                0110 ; interrupt on the overflow of Timer3
                0111 ;
0033 B0FF       0112            MOVLW   0xFF            ;
0034 0117       0113            MOVWF   T3PRH           ;
0035 0116       0114            MOVWF   T3PRL           ;
                0115 ;
                0116 ; the timer should be started and interrupts enabled.
                0117 ;
0036 B803       0118            MOVLB   3               ; Select register Bank 3
0037 8217       0119            BSF     TCON2,2         ; Turn on timer 3.
0038 8307       0120            BSF     INTSTA,3        ; Turn on Peripheral Interrupts
0039 B801       0121            MOVLB   1               ; Select register Bank 1
003A B048       0122            MOVLW   0x48            ; Enable Caputure 2 and Timer3
003B 0117       0123            MOVWF   PIE             ;    Interrupts (when GLINTD = 0)
                0124 ;
                0125 ; This is where you would do the things you wanted to do.
                0126 ; this example will only loop waiting for the interrupts.
                0127 ;
003C 8C06       0128 WAIT       BCF     CPUSTA,4        ; Enable ALL interrupts
003D C03C       0129            GOTO    WAIT            ; Loop here waiting for a timer
                0130                                    ;   Interrupt
```


© 1993 Microchip Technology Inc.

```
                    0132 ;
                    0133 ; The interrupt routine for any peripheral interrupt, This routine
                    0134 ; only deals with Timer3 (T3) interrupts.
                    0135 ;
                    0136 ; Time required to execute interrupt routine. Not including
                    0137 ; interrupt latency (time to enter into the interrupt routine)
                    0138 ;
                    0139 ;         case1 - only T3 overflow              = 12 cycles
                    0140 ;         case2 - 1st capture                   = 14 cycles
                    0141 ;         case3 - 2nd capture                   = 30 cycles
                    0142 ;         case4 - T3 overflow and 1st capture   = 34 cycles
                    0143 ;         case5 - T3 overflow and 2nd capture   = 50 cycles
                    0144 ;
                    0145 ;
003E B801           0146 PER_INT MOVLB   1               ; Select register Bank 1
003F 9E16           0147         BTFSC   PIR,6           ; Did T3 overflow?
                    0148                                 ;   If not skip next Instruction
0040 C055           0149         GOTO    T3OVFL          ; Inc overflow cntr and clear flag
0041 9316           0150 CK_CAP  BTFSS   PIR,3           ; Did the RB1/CAP2 pin cause an
                    0151                                 ;   interrupt?
0042 0005           0152         RETFIE                  ; No RB1/CAP2 interrupt,
                    0153                                 ;   Return from Interrupt
                    0154 ;
                    0155 ; This potion of the code takes the 1st capture and stores its
                    0156 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                    0157 ; is take, its value is stored in register pair IC2BH:IC2BL.
                    0158 ; A 16-bit subtract is performed, with the final 24-bit result
                    0159 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                    0160 ; be correct after the next capture occurs (IC2BH:IC2BL will
                    0161 ; change), so the main routine must utilize this value before
                    0162 ; it changes.
                    0163 ;
0043 8B16           0164 CAPTURE BCF PIR,3               ; Clear Capture2 interrupt flag
0044 B803           0165         MOVLB   3               ; Select register Bank 3
0045 9826           0166         BTFSC   FLAG_REG,0      ; 1st or 2nd capture2?
0046 C04B           0167         GOTO    CAP2            ; It was the 2nd Capture
0047 5424           0168 CAP1    MOVPF   CA2L,IC2AL      ; Move the captured value to
0048 5523           0169         MOVPF   CA2H,IC2AH      ;   temporary registers
0049 8026           0170         BSF     FLAG_REG,0      ; Have 1st capture2
004A 0005           0171         RETFIE                  ; Return from Interrupt
                    0172                                 ;

004B 5422           0174 CAP2    MOVPF   CA2L,IC2BL      ; Move the captured value to
004C 5521           0175         MOVPF   CA2H,IC2BH      ;   temporary registers
                    0176                                 ;   (to pervent being overwritten)
                    0177                                 ;
004D E061           0178         CALL    SUB16           ; Call routine which subtracts
                    0179                                 ;   2 16-bit numbers.
004E 9926           0180         BTFSC   FLAG_REG,1      ; Underflow during SUB16?
004F 0725           0181         DECF    T3OFLCNTR,1     ; Since underflow, decrement the
                    0182                                 ;   overflow counter.
0050 2926           0183         CLRF    FLAG_REG,1      ; Clear the flag bits for
                    0184                                 ;   underflow and 2nd capture2
0051 6A25           0185         MOVFP   T3OFLCNTR,W     ; Store the T3 input capture
0052 4A20           0186         MOVPF   W,IC2OF         ;   overflow value in IC2OF
0053 2825           0187         CLRF    T3OFLCNTR,0     ; Clear the Data register which
                    0188                                 ;   counts how many times Timer 3
                    0189                                 ;   overflows.
0054 0005           0190         RETFIE                  ; Return from interrupt
                    0191 ;
                    0192 ; When Timer 3 has overflowed, the overflow counter only should
                    0193 ; be incremented when the overflow occurs after a capture 1
                    0194 ; but before the capture 2. The 4 possible cases when entering
                    0195 ; the T3OVFL section of the PER_INT routine are as follows:
                    0196 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                    0197 ;           for Capture 1 to occur). Do Not increment counter
                    0198 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
                    0199 ;           for Capture 2 to occur). Increment counter
                    0200 ;   Case 3: T3 Overflow happened after Capture. Do Not
```


© 1993 Microchip Technology Inc.

```
                0201 ;           increment overflow counter
                0202 ;   Case 4: T3 Overflow occured before Capture 2 and FLAG_REG.0 = 1
                0203 ;           (waiting for Capture 2 to occur). Increment counter
                0204
                0205 ;
0055 8E16       0206 T3OVFL     BCF PIR,6               ; Clear Overflow interrupt flag
0056 9316       0207            BTFSS   PIR,3           ; Did the RB1/CAP2 pin also
                0208                                    ;   cause an interrupt?
0057 C05E       0209            GOTO    FR0             ; No, Check if between 1st
                0210                                    ;   and 2nd capture
0058 B803       0211            MOVLB   3               ; Bank 3
0059 280A       0212            CLRF    W,0             ; W = 0
005A 3115       0213            CPFSEQ  CA2H            ; if CA2H = 0, overflow happened
005B C05E       0214            GOTO    FR0             ;   first, must check FLAG_REG
                0215                                    ;   bit 0
005C B801       0216            MOVLB   1               ; Back to bank 1
005D C043       0217            GOTO    CAPTURE         ; Capture happened first, do NOT
                0218                                    ;   Increment overflow counter
                0219                                    ;   and do capture routine
005E 9826       0220 FR0        BTFSC   FLAG_REG,0      ; Between Capture 1 and Capture 2?
005F 1525       0221            INCF    T3OFLCNTR,1     ; Yes, Inc. the overflow counter
0060 0005       0222            RETFIE                  ; Return from overflow interrupt
                0223 ;
0061 6A24       0224 SUB16      MOVFP   IC2AL,W         ; Do the 16-bit subtraction
0062 0522       0225            SUBWF   IC2BL,1         ;
0063 6A23       0226            MOVFP   IC2AH,W         ;
0064 0321       0227            SUBWFB  IC2BH,1         ;
0065 9004       0228            BTFSS   ALUSTA,0        ; Is the result pos. or neg. ?
0066 8126       0229            BSF     FLAG_REG,1      ; neg., Set the underflow flag
0067 0002       0230            RETURN                  ; Return from the subroutine
                0232 ;
                0233 ; Other Interrupt routines. (Not utilized in this example)
                0234 ;
0068 0005       0235 EXT_INT    RETFIE                  ; RA0/INT interrupt routine
                0236                                    ;   (NOT used in this program)
0069 0005       0237 RTCCINT    RETFIE                  ; RTCC overflow interrupt routine
                0238                                    ;   (NOT used in this program)
006A 0005       0239 RT_INT     RETFIE                  ; RA1/RT interrupt routine
                0240                                    ;   (NOT used in this program)
                0241                                    ;
006B C028       0242 SRESET     GOTO    START           ; If program became lost, goto
                0243                                    ;   START and reinitalize.
                0244 ;
                0245 ;
                0246 ; When the executed address is NOT in the program range, the
                0247 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                0248 ; this location you could branch to a routine to recover or
                0249 ; shut down from the invalid program execution.
                0250 ;
                0251            ORG     END_OF_PROG_MEM ;
07FF C06B       0252            GOTO    SRESET          ; The program has lost it's mind,
                0253                                    ;   do a system reset
                0254                    END
                0255
```


© 1993 Microchip Technology Inc.

APPENDIX C - PULSE WIDTH MEASUREMENT EXAMPLE CODE

```
                    0001         LIST    p=17C42, c=132
                    0002 ; This is the basic outline for a program that can determine the
                    0003 ; Pulse Width of an input, via input capture. This routine uses an
                    0004 ; 8-bit register to count the times that timer3 overflowed. At the
                    0005 ; Max crystal frequency of 16 MHz, this gives an overflow time of
                    0006 ; (2**16)(256 + 1)(250 nS) > 4.21 sec or a frequncy < 0.25 Hz. If
                    0007 ; measurement of longer time intervals is required, the overflow
                    0008 ; counter could be extended to 16 (or more) bits.
                    0009 ;
                    0010 ;  Program:   PW02.ASM
                    0011 ;  Revision:  1.0
                    0012 ;
                    0013 ; Do the EQUate table
                    0014 ;
0020                0015 IC2OF           EQU     0x20            ; T3 overflow register
0021                0016 IC2BH           EQU     0x21            ; T3 ICA2 MSB register (2nd Cap)
0022                0017 IC2BL           EQU     0x22            ; T3 ICA2 LSB register
0023                0018 IC2AH           EQU     0x23            ; T3 ICB2 MSB register (1st Cap)
0024                0019 IC2AL           EQU     0x24            ; T3 ICB2 LSB register
0025                0020 T3OFLCNTR       EQU 0x25                ; Temperay T3 overflow register
                    0021 ;
0026                0022 FLAG_REG        EQU 0x26                ; Register that has the Flag bits
                    0023 ;
                    0024 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                    0025 ;                    -   -   -   -   -   -  UFL CAP1
                    0026 ;    CAP1 = 0,  1st Capture
                    0027 ;         = 1,  2nd Capture
                    0028 ;
                    0029 ;    UFL = 0,  No Underflow
                    0030 ;        = 1,  Underflow during subtract
                    0031 ;
07FF                0032 END_OF_PROG_MEM         EQU             0x07FF
                    0033 ;
0004                0034 ALUSTA          EQU     0x04
0006                0035 CPUSTA          EQU     0x06
0007                0036 INTSTA          EQU     0x07
000A                0037 W               EQU     0x0A
                    0038                                         ;
0012                0039 PORTB           EQU     0x12            ; Bank 0
                    0040                                         ;
0016                0041 PIR             EQU     0x16            ; Bank 1
0017                0042 PIE             EQU     0x17
                    0043                                         ;
0012                0044 TMR3L           EQU     0x12            ; Bank 2
```


© 1993 Microchip Technology Inc.

```
0013                0045 TMR3H          EQU     0x13
0016                0046 T3PRL          EQU     0x16
0017                0047 T3PRH          EQU     0x17
                    0048                                ;
0014                0049 CA2L           EQU     0x14        ; Bank 3
0015                0050 CA2H           EQU     0x15
0016                0051 TCON1          EQU     0x16
0017                0052 TCON2          EQU     0x17

0054 ;
                    0055                ORG     0x0000      ; Origin for the RESET vector
0000 C028           0056                GOTO    START       ; On reset, go to the start of
                    0057                                    ;   the program
                    0058                ORG     0x0008      ; Origin for the external RA0/INT
                    0059                                    ;   interrupt vector
0008 C072           0060                GOTO    EXT_INT     ; Goto the ext. interrupt
                    0061                                    ;   on RA0/INT routine
                    0062                ORG     0x0010      ; Origin for the RTCC
                    0063                                    ;   overflow interrupt vector
0010 C073           0064                GOTO    RTCCINT     ; Goto the RTCC overflow interrupt
                    0065                                    ;   routine
                    0066                ORG     0x0018      ; Origin for the external
                    0067                                    ;   RA1/RT interrupt vector
0018 C074           0068                GOTO    RT_INT      ; Goto the ext. interrupt on
                    0069                                    ;   RA1/RT routine
                    0070                ORG     0x0020      ; Origin for the interrupt vector
                    0071                                    ;   of any enabled peripheral
0020 C03E           0072                GOTO    PER_INT     ; Goto the interrupt from a
                    0073                                    ;   peripheral routine
                    0074                ORG     0x0028      ; Origin for the top of
                    0075                                    ;   program memory


MPAS0028 8406       0077 START          BSF     CPUSTA,4    ; Disable ALL interrupts via the
                    0078                                    ;   Global Interrupt Disable
                    0079                                    ;   (GLINTD) bit.
                    0080                                    ;
                    0081 MAIN                               ; Place Main program here
0029 B803           0082                MOVLB   3           ; Select register Bank 3
002A 2817           0083                CLRF    TCON2,0     ; Stop the timers, Single Capture
002B B070           0084                MOVLW   0x070       ; Initalize TCON1 so that
002C 0116           0085                MOVWF   TCON1       ;   T1 (8-bit), T2 (8-bit),
                    0086                                    ;   and T3 run off the internal
                    0087                                    ;   system clock. Capture2
                    0088                                    ;   captures on the rising edge.
                    0089 ;
                    0090 ; Initalize Timer 3, load the timer with the number of cycles that
```

© 1993 Microchip Technology Inc.

```
                        0091 ; the device executes (from RESET) before the timer is turned on
                        0092 ; Therefore the offset is required due to software overhead.
                        0093 ;
002D B802               0094                MOVLB   2                       ; Select register Bank 2
002E 280A               0095                CLRF    W,0                     ; Clear the W register
002F 0126               0096                MOVWF   FLAG_REG                ; Initalize to 0
0030 0113               0097                MOVWF   TMR3H                   ; Timer3 MSB = 0
0031 B013               0098                MOVLW   0x13                    ; Timer3 LSB = Offset
0032 0112               0099                MOVWF   TMR3L                   ;
                        0100 ;
                        0101 ; Load the Timer 3 period register with 0xFFFF, which will give an
                        0102 ; interrupt on the overflow of Timer3
                        0103 ;
0033 B0FF               0104                MOVLW   0xFF                    ;
0034 0117               0105                MOVWF   T3PRH                   ;
0035 0116               0106                MOVWF   T3PRL                   ;
                        0107 ;
                        0108 ; the timer should be started and interrupts enabled.
                        0109 ;
0036 B803               0110                MOVLB   3                       ; Select register Bank 3
0037 8217               0111                BSF     TCON2,2                 ; Turn on timer 3.
0038 8307               0112                BSF     INTSTA,3                ; Turn on Peripheral Interrupts
0039 B801               0113                MOVLB   1                       ; Select register Bank 1
003A B048               0114                MOVLW   0x48                    ; Enable Caputure 2 and Timer3
003B 0117               0115                MOVWF   PIE                     ;    Interrupts (when GLINTD = 0)
                        0116 ;
                        0117 ; This is where you would do the things you wanted to do.
                        0118 ; this example will only loop waiting for the interrupts.
                        0119 ;
003C 8C06               0120 WAIT    BCF            CPUSTA,4                ; Enable ALL interrupts
003D C03C               0121         GOTO   WAIT                            ; Loop here waiting for a timer
                        0122                                                ;   Interrupt


                        0124 ;
                        0125 ; The interrupt routine for any peripheral interrupt, This routine
                        0126 ; only deals with Timer3 (T3) interrupts.
                        0127 ;
                        0128 ; Time required to execute interrupt routine. Not including
                        0129 ; interrupt latency (time to enter into the interrupt routine)
                        0130 ;
                        0131 ;   case1 - only T3 overflow            = 12 cycles
                        0132 ;   case2 - 1st capture                 = 20 cycles
                        0133 ;   case3 - 2nd capture                 = 34 cycles
                        0134 ;   case4 - T3 overflow and 1st capture  = 32 cycles
                        0135 ;   case5 - T3 overflow and 2nd capture  = 44 cycles
                        0136 ;
```


© 1993 Microchip Technology Inc.

```
                0137 ;
003E B801       0138 PER_INT        MOVLB   1               ; Select register Bank 1
003F 9E16       0139                BTFSC   PIR,6           ; Did T3 overflow?
                0140                                        ;   If not skip next Instruction
0040 C05A       0141                GOTO    T3OVFL          ; Inc overflow cntr and clear flag
0041 9316       0142 CK_CAP         BTFSS   PIR,3           ; Did the RB1/CAP2 pin cause an
                0143                                        ;   interrupt?
0042 0005       0144                RETFIE                  ; No RB1/CAP2 interrupt,
                0145                                        ;   Return from Interrupt
                0146
                0147 ;
                0148 ; This potion of the code takes the 1st capture and stores its
                0149 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                0150 ; is take, its value is stored in register pair IC2BH:IC2BL.
                0151 ; A 16-bit subtract is performed, with the final 24-bit result
                0152 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                0153 ; be correct after the next capture occurs (IC2BH:IC2BL will
                0154 ; change), so the main routine must utilize this value before
                0155 ; it changes.
                0156 ;
0043 B803       0157 CAPTURE        MOVLB   3               ; Select register Bank 3
0044 9826       0158                BTFSC   FLAG_REG,0      ; Capture on rising or falling edge?
0045 C04B       0159                GOTO    FALLING         ; It was the 2nd Capture
0046 5424       0160 RISING         MOVPF   CA2L,IC2AL      ; Move the captured value to
0047 5523       0161                MOVPF   CA2H,IC2AH      ;   temporary registers
0048 8026       0162                BSF     FLAG_REG,0      ; Set flag for 1st capture
0049 8E16       0163                BCF     TCON1,6         ; Change edge from rising
                0164                                        ;   to falling
004A C055       0165                GOTO    FALSE_C         ;** With the changing of the capture
                0166                                        ;** edge, we have a false capture
                0167                                        ;

004B 5422       0169 FALLING        MOVPF   CA2L,IC2BL      ; Move the captured value to
004C 5521       0170                MOVPF   CA2H,IC2BH      ;   temporary registers
                0171                                        ;   (to pervent being overwritten)
                0172                                        ;
004D E06B       0173                CALL    SUB16           ; Call routine which subtracts
                0174                                        ;   2 16-bit numbers.
004E 9926       0175                BTFSC   FLAG_REG,1      ; Underflow during SUB16?
004F 0725       0176                DECF    T3OFLCNTR,1     ; Since underflow, decrement the
                0177                                        ;   overflow counter.
0050 2926       0178                CLRF    FLAG_REG,1      ; Clear the flag bits for
                0179                                        ;   underflow and 2nd capture2
0051 6A25       0180                MOVFP   T3OFLCNTR,W     ; Store the T3 input capture
0052 4A20       0181                MOVPF   W,IC2OF         ;   overflow value in IC2OF
0053 2825       0182                CLRF    T3OFLCNTR,0     ; Clear the Data register which
                0183                                        ;   counts how many times Timer 3
                0184                                        ;   overflows.
```


© 1993 Microchip Technology Inc.

```
0054 8616          0185                BSF             TCON1,6         ; Change edge from falling
                   0186                                                ;   to rising
                   0187 ;
                   0188 ; Note when you change the edge of the capture, an additional capture
                   0189 ; is generated. The capture register must be read before the capture
                   0190 ; flag is cleared.
                   0191 ;
0055 550A          0192 FALSE_C        MOVPF   CA2H,W                  ; Dummy read of Capture 2
0056 540A          0193                MOVPF   CA2L,W                  ;
                   0194 ;
0057 B801          0195                MOVLB   1                       ; Select register Bank 1
0058 8B16          0196                BCF     PIR,3                   ; Clear Capture2 Interrupt flag
0059 0005          0197                RETFIE                          ; Return from interrupt, wait for
                   0198                                                ;   T3 overflow or falling edge
                   0199                                                ;   capture
                   0200                                                ;

                   0202 ;
                   0203 ; When Timer 3 has overflowed, the overflow counter only should
                   0204 ; be incremented when the overflow occurs after a capture 1
                   0205 ; but before the capture 2. The 6 possible cases when entering
                   0206 ; the T3OVFL section of the PER_INT routine are as follows:
                   0207 ;
                   0208 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                   0209 ;           for Capture 1 to occur). Do Not increment counter
                   0210 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
                   0211 ;           for Capture 2 to occur). Increment counter
                   0212 ;   Case 3: T3 Overflow, Then Capture1 happened. Do Not
                   0213 ;           increment overflow counter
                   0214 ;   Case 4: T3 Overflow, Then Capture 2 happened
                   0215 ;           Increment counter
                   0216 ;   Case 5: Capture1, Then T3 Overflow happened
                   0217 ;           Increment counter
                   0218 ;   Case 6: Capture2, Then T3 Overflow happened. Do Not
                   0219 ;           Increment counter
                   0220 ;
005A 8E16          0221 T3OVFL         BCF     PIR,6                   ; Clear Overflow interrupt flag
005B 9316          0222                BTFSS   PIR,3                   ; Did the RB1/CAP2 pin also
                   0223                                                ;   cause an interrupt?
005C C068          0224                GOTO    FR0                     ; No, only overflow interrupt
005D B803          0225                MOVLB   3                       ; Bank 3
005E 280A          0226                CLRF    W,0                     ; W = 0
005F 9826          0227                BTFSC   FLAG_REG,0              ; T3 overflow with Capture 1
                   0228                                                ;   or Capture 2?
0060 C064          0229                GOTO    OF_C1                   ; Overflow with Capture 1
0061 3115          0230 OF_C2          CPFSEQ  CA2H                    ; if CA2H = 0, overflow happened
                   0231                                                ;   first
0062 1525          0232                INCF    T3OFLCNTR,1             ; Increment counter
0063 C043          0233                GOTO    CAPTURE                 ; Do capture routine
```


© 1993 Microchip Technology Inc.

```
0064 3115          0234 OF_C1          CPFSEQ  CA2H                ; if CA2H = 0, overflow happened
                   0235                                            ;   first
0065 C043          0236                GOTO    CAPTURE             ; Capture happened first, do NOT
                   0237                                            ;   Increment overflow counter
                   0238                                            ;   and do capture routine
0066 1525          0239                INCF    T3OFLCNTR,1         ; Yes, Inc. the overflow counter
0067 C043          0240                GOTO    CAPTURE             ; Do capture routine
                   0241 ;
                   0242 ; Only increment overflow counter if between 1st and 2nd capture
                   0243 ;
0068 9826          0244 FR0            BTFSC   FLAG_REG,0          ; Between Capture 1 and Capture 2?
0069 1525          0245                INCF    T3OFLCNTR,1         ; Yes, Inc. the overflow counter
006A 0005          0246                RETFIE                      ; Return from overflow interrupt
                   0247 ;
006B 6A24          0248 SUB16          MOVFP   IC2AL,W             ; Do the 16-bit subtraction
006C 0522          0249                SUBWF   IC2BL,1             ;
006D 6A23          0250                MOVFP   IC2AH,W             ;
006E 0321          0251                SUBWFB  IC2BH,1             ;
006F 9004          0252                BTFSS   ALUSTA,0            ; Is the result pos. or neg. ?
0070 8126          0253                BSF             FLAG_REG,1  ; neg., Set the underflow flag
0071 0002          0254                RETURN                      ; Return from the subroutine


                   0256 ;
                   0257 ; Other Interrupt routines. (Not utilized in this example)
                   0258 ;
0072 0005          0259 EXT_INT        RETFIE                      ; RA0/INT interrupt routine
                   0260                                            ;   (NOT used in this program)
0073 0005          0261 RTCCINT        RETFIE                      ; RTCC overflow interrupt routine
                   0262                                            ;   (NOT used in this program)
0074 0005          0263 RT_INT         RETFIE                      ; RA1/RT interrupt routine
                   0264                                            ;   (NOT used in this program)
                   0265                                            ;
0075 C028          0266 SRESET         GOTO    START               ; If program became lost, goto
                   0267                                            ;   START and reinitalize.
                   0268 ;
                   0269 ;
                   0270 ; When the executed address is NOT in the program range, the
                   0271 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                   0272 ; this location you could branch to a routine to recover or
                   0273 ; shut down from the invalid program execution.
                   0274 ;
                   0275                ORG     END_OF_PROG_MEM ;
07FF C075          0276                GOTO    SRESET              ; The program has lost it's mind,
                   0277                                                    ;   do a system reset
                   0278                END
                   0279
```


© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

MICROCHIP

AN547

Serial Port Utilities

INTRODUCTION

PIC17C42 has an on chip high speed Universal Synchronous Asynchronous Receiver Transmitter (USART). The serial port can be configured to operate either in full-duplex asynchronous mode or half duplex synchronous mode. The serial port has a dedicated 8-bit baud rate generator. Either 8- or 9-bits can be transmitted/received.

This application note provides information on using the serial port, parity generation, serial port expansion, RS-232 interface, I/O port expansion using synchronous mode of serial port.

SERIAL PORT USAGE

Brief code to setup serial port , receive and transmit data is given below. Small sections of code for both asynchronous and synchronous mode are given.

EXAMPLE 1

Asynchronous Mode Setup

Asynchronous mode set-up requires selection of 8/9-bits of data transfer, baud rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{Input_Clk_Freq}{64 * Baud_Rate} - 1$$

For example, to select a baud rate of 9600 bits/sec with input clock frequency of 16 MHz, SPBRG is computed from the above equation to be 25. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet) :

```
SPBRG   :   25              :   9600 baud @ 16 MHz input clock
TXSTA   :   00100000  (20h) :   8-bit transmission, async mode
RCSTA   :   10010000  (90h) :   8-bit reception, enable serial port, enable reception

;*********************************************************************
;       Sample Code For Asynchronous Mode Serial Port Setup
;*********************************************************************

#define ClkFreq     16000000                    ; input clock frequency = 16 MHz
#define baud(X)     ((10*ClkFreq/(64*X))+5)/10 - 1
#define TXSTA_INIT  0xB0
#define RCTSA_INIT  0x90

    #include        "17C42.h"                   ; file containing the Register Definitions

    Setup_Async_Mode
                    movlb   0                   ; SPBRG, TXSTA & RCSTA are in bank 0
                    movlw   baud(9600)          ; equals 25 for 9600 baud
                    movwf   SPBRG               ; baud rate generator is reset & initialized
                    movlw   TXSTA_INIT
                    movwf   TXSTA               ; 8-bit transmission, async mode
                    movlw   RCSTA_INIT
                    movwf   RCSTA               ; 8-bit reception, enable serial port,
                                                ; enable reception
                    return
    ;*************************************************************************
```

© 1993 Microchip Technology Inc.

Synchronous Mode Setup

Synchronous mode setup requires selection of 8/9-bits of data transfer, bit rate, setting the baud rate generator and configuring the TXSTA & RCSTA control registers. The baud rate generator is set by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{Input_Clk_Freq}{4 * Baud_Rate} - 1$$

For example, to select a bit rate of 1 MHz with input clock frequency of 16 Mhz, SPBRG is computed from the above equation to be 3. Once the Baud Rate Generator is setup, it is necessary to configure the TXSTA & RCSTA control registers as follows (please refer to the data sheet) :

EXAMPLE 2

```
;
;       SPBRG   :       3               :  1 Mbits/sec @ 16 Mhz input clock
;       TXSTA   :       10110000  (B0h) :  8-bit transmission, Sync mode (MASTER)
;       RCSTA   :       10010000  (90h) :  8-bit reception, enable serial port, continuous
;                                          reception
;
;***********************************************************************
;               Sample Code For Synchronous Mode (MASTER) Serial Port Setup
;***********************************************************************
#define ClkFreq         16000000        ; input clock frequency = 16 Mhz
#define baud(X)         ((10*ClkFreq/(4*X))+5)/10 - 1
#define TXSTA_INIT      0xB0
#define RCTSA_INIT      0x90

        #include                "17C42.h"       ; file containing the Register Definitions

Setup_Sync_Master_Mode
                movlb   0                       ; SPBRG, TXSTA & RCSTA are in bank 0
                movlw   baud(1000000)           ; equals 3 for 1 Mbits/sec
                movwf   SPBRG                   ; baud rate generator is reset & initialized
                movlw   TXSTA_INIT
                movwf   TXSTA                   ; 8-bit transmission, async mode
                movlw   RCSTA_INIT
                movwf   RCSTA                   ; 8-bit reception, enable serial port,
                                                ; enable reception
                return
;***********************************************************************
```


© 1993 Microchip Technology Inc.

Receiving Data (Software Polling)

The sample code provides a way to read the received serial data by software polling (with no serial port interrupts). This applies to both asynchronous and synchronous mode. Software polling is done by checking the RBFL bit (PIR<0>). If this bit is set it means that a word has been received (8 bits are in RCREG and the 9th bit in RCSTA<0>).

EXAMPLE 3

```
;*****************************************************************
;             Return The 8-bit received Data By Software Polling
;       The received data is returned in location SerInData
;*****************************************************************
Get_Serial_Data_Poll
                movlb   1               ; PIR is in bank 1
PollRcv         btfss   PIR,0           ; chech the RBFL bit
                goto    PollRcv         ; loop until char received, assume WDT is off
                movlb   0               ; RCREG is in bank 0
                movpf   RCREG,SerInData
                return                  ; Received 8-bits are in SerInData
;*****************************************************************
```

Transmitting Data (Software Polling)

The sample code provides a way to transmit serial data by software polling (no serial port interrupts). Software polling is done by checking the TBMT bit (PIR<0> in bank 1) to be one, indicating the transfer of TXREG to the serial shift register.

EXAMPLE 4

```
;*****************************************************************
;             Transmit 8-bit  Data By Software Polling
;       The data to be transmitted is in location SerOutData
;*****************************************************************
Send_Serial_Data_Poll
                movlb   1               ; PIR is in bank 1
PollTxmt        btfss   PIR,1           ; chech the TBMT bit of PIR register in bank1
                goto    PollTxmt        ; loop until char received, assume WDT is off
                movlb   0               ; RCREG is in bank 0
                movfp   SerOutData,TXREG
                return                  ; Received 8-bits are in SerInData
;*****************************************************************
```

© 1993 Microchip Technology Inc.

Transmitting & Receiving A Block Of Data (Interrupt Driven)

A general purpose routine which is interrupt driven that transmits and receives a block of data is provided. The reception or transmission of the block is ended when an end of block character is detected. As an example, the end of block is identified by a 0. The block of data to be transmitted is stored in the program memory and TABLRD instruction is used to transfer this example data to the file registers and serial port. The user may modify this code to a more general purpose routine that suits his application.

EXAMPLE 5

```
MPASM B0.54                                                    PAGE  1


                        ;       TITLE   'Serial Interface Routines
                                LIST P=17C42, C=80. I=ON, R=DEC

                        ;This is a short program to demonstrate how to transmit and

                        ;serial data using the PIC17C42.
                        ;
                        ;A message will be transmitted and routed right back to the

                        ;and read. The read information will be saved in an interna

                        ;
                                include "p17reg.h"




0080                    TX_BUFFER       equ     0x80
00B0                    RX_BUFFER       equ     0xB0
0020                    RXPTR           equ     0x20
0021                    TXPTR           equ     0x21
0022                    SERFLAG         equ     0x22
0023                    RTINUM          equ     0x23
0001                    RXDONE          equ     1
0000                    TXDONE          equ     0
0002                    HILOB           equ     2
                        ;
                        ;
                                ORG     0
0000 C072                       goto    start
                        ;
                                ORG     0x0010          ;vector for rtcc interrupt

                        ;rtcc_int                       ;not used here
                        ;
                                ORG     0x0020          ;vector for peripheral inte

                        perf_int
0020 C04D                       goto    service_perf    ;service the interrupts
                        ;
                                ORG     0x0030
                        ;
                        ;initialize the serial port: baud rate interrupts etc.
                        init_serial
0030 2922                       clrf    SERFLAG         ;clear all flags
0031 B800                       movlb   0               ;select bank 0
0032 B007                       movlw   0x07            ;select 9600 baud
0033 770A                       movfp   W,SPBRG         ;       /
0034 B090                       movlw   0x90            ;set up serial pins
0035 730A                       movfp   W,RCSTA         ;       /
```


© 1993 Microchip Technology Inc.

```
0036 2915                clrf    TXSTA           ;setup transmit status
0037 B801                movlb   1               ;select bank 1
0038 2916                clrf    PIR             ;clear all interrupts
0039 2917                clrf    PIE             ;clear all enables
003A 8017                bsf     PIE,RCIE        ;enable receive interrupt
003B B0B0                movlw   RX_BUFFER       ;set pointer to rx buffer
003C 4A20                movpf   W,RXPTR         ;       /
003D 2907                clrf    INTSTA          ;clear all interrupts
003E 8307                bsf     INTSTA,PEIE     ;enable peripheral ints
003F 0005                retfie
                         ;
                         ;start transmission of first two bytes
                         start_xmit
0040 B800                movlb   0               ;select bank 0
0041 8515                bsf     TXSTA,TXEN      ;enable transmit
0042 AB0A                tablrd  1,1,W           ;load latch
0043 A216                tlrd    1,TXREG         ;load high byte
0044 B801                movlb   1               ;select bank 1
                         empty_chk
0045 9116                btfss   PIR,TBMT        ;TXBUF empty?
0046 C045                goto    empty_chk       ;no then keep checking
0047 B800                movlb   0               ;select bank 0
0048 A916                tablrd  0,1,TXREG       ;load lo byte
0049 B801                movlb   1               ;select bank 1
004A 8117                bsf     PIE,TXIE        ;enable transmit interrupts

004B 8222                bsf     SERFLAG,HILOB   ;set up next for high byte

004C 0002                return
                         ;
                         ;
                         ;
                         service_perf
                         ;check for transmit or receive interrupts only
004D 9816                btfsc   PIR,RBFL        ;RX buffer full?
004E C062                goto    service_recv    ;yes then service
004F 9116                btfss   PIR,TBMT        ;TX buffer empty?
0050 C060                goto    exit_perf       ;no, ignore other int.
                         service_xmt
0051 9822                btfsc   SERFLAG,TXDONE  ;all done?
0052 C060                goto    exit_perf       ;yes then quit
0053 9A22                btfsc   SERFLAG,HILOB   ;if clr, do low byte
0054 C057                goto    rd_hi           ;else read high byte
0055 A90A                tablrd  0,1,W           ;read lo
0056 C058                goto    sx_cont         ;continue
                         rd_hi
0057 A20A                tlrd    1,W             ;read high byte
                         sx_cont
0058 3A22                btg     SERFLAG,HILOB   ;toggle flag
0059 B800                movlb   0               ;bsr=0
005A 4A16                movpf   W,TXREG         ;load tx reg
005B 330A                tstfsz  W               ;last byte?
005C C060                goto    exit_perf       ;no then cont
                         end_xmt                 ;else end transmit
005D B801                movlb   1               ;select bank 1
005E 8917                bcf     PIE,TXIE        ;disable tx interrupt
005F 8022                bsf     SERFLAG,TXDONE  ;set done flag
                         exit_perf
0060 8F07                bcf     INTSTA,PEIR     ;clear peripheral int
0061 0005                retfie
                         ;
                         service_recv
0062 9922                btfsc   SERFLAG,RXDONE  ;RX complete?
0063 C060                goto    exit_perf       ;exit int
0064 6120                movfp   RXPTR,FSR0      ;get pointer
0065 B800                movlb   0               ;select bank 0
0066 6014                movfp   RCREG,F0        ;load received value
0067 290A                clrf    W               ;clr W
0068 3200                cpfsgt  F0              ;value = 0?
```

4

© 1993 Microchip Technology Inc.

```
0069 C06D                       goto    end_recv        ;yes then end
006A 1501                       incf    FSR0            ;inc pointer
006B 4120                       movpf   FSR0,RXPTR      ;save pointer
006C C060                       goto    exit_perf       ;return from int
                        end_recv
006D 8122                       bsf     SERFLAG,RXDONE  ;set flag
006E 2907                       clrf    INTSTA          ;clear all int
006F B801                       movlb   1               ;select bank 1
0070 8817                       bcf     PIE,RCIE        ;disable rx interrupts
0071 C060                       goto    exit_perf       ;return
                        ;
                        start
0072 2909                       clrf    FSR1            ;assign FSR1 as S.P.
0073 0709                       decf    FSR1            ;       /
0074 B020                       movlw   0x20            ;clear ram space
0075 610A                       movfp   W,FSR0          ;do indirect addressing
                        start1
0076 2900                       clrf    F0              ;clear ram
0077 1F01                       incfsz  FSR0            ;inc and skip if done
0078 C076                       goto    start1
0079 E030                       call    init_serial     ;initialize serial port
007A B000                       movlw   HIGH MESSAGE        ;load table pointer
007B 4A0D                       movpf   W,TBLPTRL       ;       /
007C B001                       movlw   LOW MESSAGE     ;     /
007D 4A0E                       movpf   W,TBLPTRH       ;     /
007E E040                       call    start_xmit      ;start transmission
                        chk_end
007F 9122                       btfss   SERFLAG,RXDONE  ;receive all?
0080 C07F                       goto    chk_end         ;no then keep checking
                        ;
0081 C081               loop    goto    loop            ;spin wheel
                        ;
                                ORG     0x100
                        MESSAGE
0100 5468 6520 636F             DATA    "The code is: Tea for the Tillerman"
0103 6465 2069 733A
0106 2054 6561 2066
0109 6F72 2074 6865
010C 2054 696C 6C65
010F 726D 616E
0111 0000                       DATA    0
                        ;
                        ;
                                END


Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

PARITY GENERATION

Since the serial port of PIC17C42 does not have an on chip parity generator, parity is generated using software. It takes only 10 program memory words and executes in 10 instruction cycles to generate parity. Since the serial port of PIC17C42 can operate in a 9-bit mode, the parity bit can be generated in software and transmitted as the 9th bit or be compared with the received 9th bit.

In case of transmission, set TX8/9 to 1 (< 6> of TXSTA) to enable 9-bit transmission and write the computed parity bit to TXD8 (TXSTA<0>). The 9th bit (parity bit) must be written prior to writing the 8 data bits to TXREG.

In case of reception, first of all enable 9-bit reception by setting RC8/9 to 1 (RCSTA<6>). Upon successful reception, the 9 bit is received in RCD8 (RCSTA<0>). Parity of the 8 bits of received data is computed using the routine listed below and compared with the 9-bit received.

EXAMPLE 6

```
;*****************************************************************************
;   Generate Parity Bit for the 8 bit register 'data'
;   The parity bit is stored in Bit 0 of 'parity'
;
;*****************************************************************************

#define ODD_PARITY  FALSE

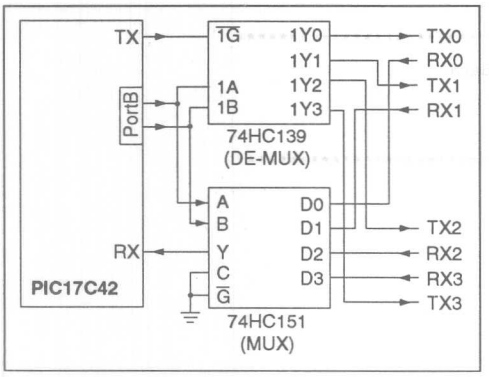
    swapf   data,w
    xorwf   data,w
    movwf   parity
    rrncf   parity
    rrncf   parity
    xorwf   parity,w
    andlw   0x03
    addlw   0x01
    rrncf   wreg
    movwf   parity
 #if ODD_PARITY
    btg parity,0
 #endif
```

4

SERIAL PORT EXPANSION

The PIC17C42 has only one serial port. For applications that require the PIC17C42 to communicate with multiple serial ports, a scheme the multiplexes and demultiplexes the RX and TX pins is provided below. This method is suited only if no more than one UART is needed at any one time. This is the case in many applications where the microcontroller drives several outputs devices serially. Figure 1 shown below suggests a way to expand the on-chip serial port to 4 serial ports. To use the scheme as shown in Figure 1, The PIC17C42 must select the desired serial port by appropriately setting the two pins of PORT-B. The same scheme may be used to further expand the serial ports by using more I/O Ports.

FIGURE 1 - MULTIPLEXING THE ON-CHIP UART



RS-232 INTERFACE

Two circuits are provided to interface the CMOS levels of PIC17C42 to RS-232 levels. Figure 2 provides interface to MAX232 (MAXIM's RS-232 Driver/Receiver) with a single +5V power supply. Figure 3 provides a low cost 2 chip solution for RS-232 level translation using a single +5V supply (Note that V- of MC14C88 is connected to DTR of RS-232 Interface. By asserting DTR to low, V- gets the negative voltage from the RS-232 line). An alternative single chip low cost solution is provided in Figure 4. However 3 voltage sources (+5, +12, -12) are necessary.

FIGURE 2 - RS-232 INTERFACE TO MAX232

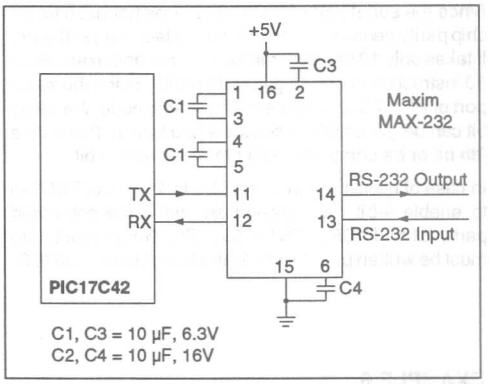


FIGURE 3 - LOW COST 2 CHIP SOLUTION USING SINGLE POWER SOURCE

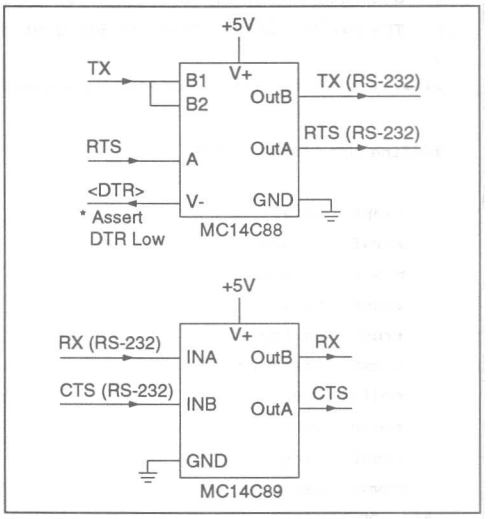
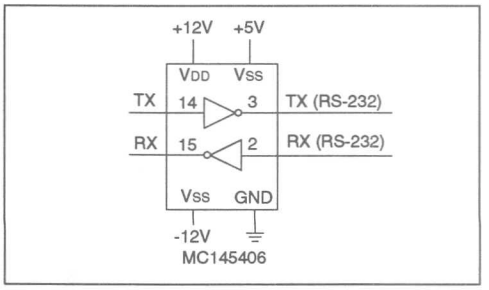


FIGURE 4 - LOW COST SINGLE CHIP SOLUTION USING 3 POWER SOURCES



© 1993 Microchip Technology Inc.

Table 1 provides the summary of RS-232 and V.28 Electrical Specifications.

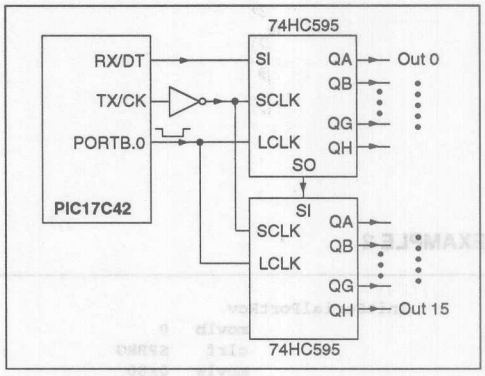
TABLE 1 - SUMMARY OF RS-232C AND V.28 ELECTRICAL SPECIFICATIONS

Parameter	Specification	Comments
Driver Output Voltage		
0 level 1 level Max. output	+5V to +15V -5V to -15V ±25V	With 3-7kΩ load with 3-7kΩ load No load
Receiver Input Thresholds (Data and clock signals)		
0 level 1 level	+3V to +25V -3V to -25V	
Receiver Thresholds RTS, DSR, DTR		
On level Off level	+3V to +25V Open circuit or -3V to -25V	Detects power Off condition at driver
Receiver input resistance	3kΩ to 7kΩ	
Driver output resistance,		
Power off condition Driver slew rate	300Ω Min. 30V/μs max.	Vout < ±2V 3kΩ < RL < 7kΩ; 0pF < CL < 2500pF
Signalling rate	Up to 20k bits/sec.	
Cable length		
	50'/15m. Recommended max. length	Longer cables permissable, if CLoad ≤ 2500pF

I/O PORT EXPANSION USING SYNCHRONOUS MODE

Although the PIC17C42 has 33 I/O pins, most of these are multiplexed with other peripheral functions. In case more I/O ports are needed, the scheme provided below expands the I/O port using the synchronous mode of serial port by serially shifting the data. Figure 5 shows a scheme to expand the output ports to 16-bits using 2 standard logic chips (74HC595). The PIC17C42's serial port is configured in synchronous mode and set to be the MASTER. Thus serial data is available on DT (pin 22) and the clock is available on CK (pin 21). The following code will transmit 16-bits serially and clock all the 16-bits at the same time.

FIGURE 5 - OUTPUT PORT EXPANSION USING SYNCHRONOUS MODE



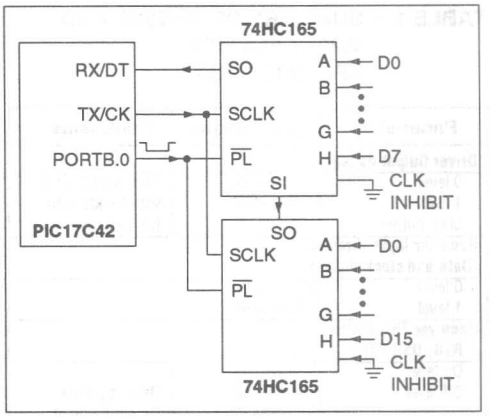
EXAMPLE 1

```
InitSerialPortTxmt
        movlb   0
        clrf    SPBRG           ; set to highest baud rate = CLKOUT = CLKIN/4
        movlw   0x80
        movwf   RCSTA           ; enable serial port
        movlw   0xB0
        movwf   TXSTA           ; 8 bit synchronous master mode
        bcf DDRB,0              ; set bit 0 of PortB as output, to be used as Latch Clk
        return
SendSerialData                  ; shift out DataLo & DataHi serially
        movlb   0
        movfp   DataLo,TXREG
        nop                     ; wait for TXREG transfer : if slower baud rate is
                                ; used check for TBMT reset
        movfp   DataHi,TXREG
wait    btfss   TXSTA,TRMT      ; wait until all 16 bits shifted out
        goto    wait
        bcf PortB,0
        bsf PortB,0             ; clock in the serial data to parallel output of HC595
        return
```

© 1993 Microchip Technology Inc.

A similar scheme as shown above may be implemented in the reverse way to expand Input Ports. For this it is necessary to have a Parallel In and Serial Out device. Using a standard logic chip (74HC165), the scheme is shown in Figure 6. In order to read the 16-bit input data, the serial port of PIC17C42 is configured to be in Synchronous Mode Reception (MASTER mode). An I/O port (PortB<0>) is used to parallel load the 16 inputs for reading serially. A sample code to read the 16 inputs is shown below.

FIGURE 6 - INPUT PORT EXPANSION USING SYNCHRONOUS MODE



Author: Amar Palacherla
Logic Products Division

EXAMPLE 2

```
InitSerialPortRcv
                movlb   0
                clrf    SPBRG           ; set to highest baud rate = CLKOUT = CLKIN/4
                movlw   0x80
                movwf   RCSTA           ; enable serial port
                movlw   0x90
                movwf   TXSTA           ; 8-bit synchronous master mode
                bcf     DDRB,0          ; bit 0 of PortB is output, to be used as Parallel
                                        ; Load
                bsf     PortB,0         ; disable parallel Load
                return
    ReadSerialData                      ; shift out DataLo & DataHi serially
                movlb   0
                bcf     PortB,0         ; Parallel Load The Inputs into 74HC165
                bsf     PortB,0         ; disable parallel Load
                bsf     RCSTA,SREN      ; enable single byte reception and wait for data
                movlb   1
          wait1 btfss   PIR,RBFL
                goto    wait1           ; check until 8-bits are received
                movlb   0
                movpf   RCREG,DataLo    ; 1st byte is read
                movlb   0
                bsf     RCSTA,SREN      ; enable another byte of reception and wait for data
                movlb   1
          wait2 btfss   PIR,RBFL
                goto    wait2           check until 8-bits are received
                movlb   0
                movpf   RCREG,DataHi    ; 2nd byte is read
                return
```


© 1993 Microchip Technology Inc.

MICROCHIP

AN544

Math Utility Routines

INTRODUCTION

This application note provides some utility math routines for Microchip's second generation of high performance 8-bit microcontroller, the PIC17C42. Three assembly language modules are provided, namely ARITH.ASM, BCD.ASM and FXP_DIV.ASM. Currently in each file the following subroutines are implemented:

ARITH.ASM

- Single precision 8 x 8 unsigned multiply
- 16 x 16 double precision multiply (signed or unsigned)
- 16 / 16 double precision divide (signed or unsigned)
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- double precision square root
- double precision numerical differentiation
- double precision numerical integration
- Pseudo Random number generation
- Gaussian distributed random number generation

BCD.ASM

- 8-bit binary to 2 digit BCD conversion
- 16-bit binary to 5 digit BCD conversion
- 5-bit BCD to 16-bit binary conversion
- 2 digit BCD addition

FXP_DIV.ASM.

The routines that are implementing in this source file are shown in Table 3.

As more routines are available, they will be added to the library. The latest routines may be obtained either through Microchip's bulletin board or by contacting your nearest Microchip sales office for a copy on a MS-DOS™ floppy.

These routines have been optimized wherever possible with a compromise between speed, RAM utilization, and code size. Some routines (multiplication and division) are provided in two forms, one optimized for speed and the other optimized for code size.

All the routines have been implemented as callable subroutines and the usage of each routine is explained below. At the end of the application note, the listing files of the above programs are given.

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 1.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of 16- bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code). These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 2.

TABLE 1: SINGLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
mpy8x8_F	speed efficient	36	36	0	used
mpy8x8_S	code efficient	13	69	1	used

© 1994 Microchip Technology Incorporated

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

The data memory organization is explained in the comment section of the code. Faster execution and code space saving can be achieved by setting "MODE_FAST equ TRUE". However, setting MODE_FAST variable to TRUE restricts that operands and the 32-bit result be in data RAM locations 0x18 and 0x1F (in this mode, MOVFP and MOVPF instructions may be used to transfer data to/from any RAM location to addresses less than 0x1F). If MODE_FAST is set to FALSE, there will be no restriction on the location of the data RAM values used in this subroutine. However, the code will be slightly slower and occupies more program memory.

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

DOUBLE PRECISION DIVISION

This routine performs a 2's complement division of two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code).

These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in Appendix C. The performance specs are shown in Table 3.

TABLE 2: DOUBLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
D_mpyF	Speed Efficient, Signed Arithmetic	204	183	1	used
D_mpyF	Speed Efficient, Unsigned Arithmetic	179	176	0	used
D_mpyS	Code Efficient, Signed Arithmetic	52	254	4	used
D_mpyS	Code Efficient, Unsigned Arithmetic	21	242	3	used

TABLE 3: FIXED POINT DIVIDE PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	Program Memory	Data Memory
16 / 8 Signed	146	135	146	5
16 / 8 Unsigned	196	156	195	4
16 / 7 Unsigned	130	130	129	4
15 / 7 Unsigned	125	125	124	4
16 / 16 Unsigned	214	187	241	7
16 / 16 Unsigned	244	180	243	6
16 / 15 Unsigned	197	182	216	6
16 / 15 Unsigned	191	177	218	6
32 / 16 Unsigned	414	363	476	9
32 / 16 Unsigned	485	459	608	9
32 / 15 Unsigned	390	359	451	8
31 / 15 Unsigned	383	353	442	8

© 1994 Microchip Technology Incorporated

DOUBLE PRECISION ADDITION AND SUBTRACTION

Two routines are provided. One performs a 2's complement addition and the other one performs a 2's complement subtraction of two 16-bit binary numbers. These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 4.

NEGATE A DOUBLE PRECISION NUMBER

These routines negate a double precision number (16-bit and 32-bit). Two routines and two macros are provided to negate a 16-bit number. The subroutines use indirect addressing mode and the macros use a direct addressing scheme. A macro is provided to negate a 32- bit number.

DOUBLE PRECISION SQUARE ROOT

Often in many applications, one needs to find the square root of a number. Of the many numerical methods available to compute the square root of a number, the Newton-Raphson method is one of the most attractive because of its fast convergence rate. In this method, the square root of number, N, is obtained as an approximate solution of

$$f(Y) = Y^2 - N = 0$$

The function f(Y) can be expanded about Y0 using the first order Taylor polynomial expansion as:

Equation 1:

$$f(Y) = f(Y_0) + (Y - Y_0)f'(Y_0) + \frac{(Y - Y_0)2f'' f(Y_0)}{2!} + \ldots.$$

If X is a root of f(Y), then f(X) = 0: Therefore,

$$f(X) = f(Y_0) + (X - Y_0)f'(Y_0) + \frac{(X - Y_0)2f'' f(Y_0)}{2!} + \ldots. = 0$$

Therefore, $f(Y_0) + (X - Y_0)f'(Y_0) = 0$

i.e., $X = Y_0 + \frac{f(Y_0)}{'f(Y_0)}$

4

TABLE 4: DOUBLE PRECISION ADDITION AND SUBTRACTION

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Dadd	4	4	0	used
Dsub	4	4	0	used

TABLE 5: NEGATE A DOUBLE PRECISION NUMBER

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Negate	7	7	0	unused
NegateAlt	7	7	0	used
NegMac	5	5	0	used
AltNegMac	5	5	0	unused
NegMac32 (32 bit)	11	11	0	used

© 1994 Microchip Technology Incorporated

Thus X is a better approximation for Y0. From the previous equation, the sequence {Xn} can be generated:

Equation 2: $X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, \quad n >= 1$

For our case, equation 2, reduces to:

Equation 3: $\frac{X_{n-1} + \frac{N}{X_{n-1}}}{2}$

The routine "Sqrt" in ARITH.ASM implements the above equation. Equation 3 requires that at first an initial approximation for the root is known. The better the initial approximation, the faster the convergence rate would be. In the "Sqrt" routine, the initial approximation root is set as N/2. This routine calls the double precision division routine (D_divS).

In the code size, the Division routine (Ddiv_S) size is not included.

BCD ROUTINES

Three routines are provided for general purpose BCD arithmetic:

a. BCD to binary conversion

b. Binary to BCD conversion

c. BCD addition

The BCD to binary conversion routine converts a 5 digit BCD code to a 16-bit binary number. The BCD addition routine adds two BCD digits directly without converting them at first to binary. Note the usage of the **"DAW"** instruction. The other two routines convert a binary number to a BCD code. The performance specs for the BCD routines is given in the Table 7 below.

NUMERICAL DIFFERENTIATION

This routine performs numerical differentiation of a sequence of data if the input sequence is assumed to be piecewise linear with no discontinuances (this is the case in most of the real world signals). Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. This routine uses the so called 3-Point formula to compute the differential of a sequence of numbers.

Given an equation f(t), its derivative is given by

$$f'(t) = \frac{df(t)}{dt}$$

The above equation can be approximated using the 3-Point formula as given below:

3-Point Formula:

$$f'(t) = \frac{df(t)}{dt} = \frac{1}{2h}\,[f(t_0 - 2h) - 4f(T_0 - h) + 3f(t_0)]$$

where t_0 is the point at which the numerical derivative is desired and "h" is the step size. The smaller the value of the step size (h), the better the approximation. In case of say, PID motor control, the step size is proportional to the time intervals at which the new sample value of the position (or speed) is obtained. Using the above equation to compute the differential, three samples are necessary (present value and the last two past values). The subroutine "Diff" is implemented so that 1/2h factor is stored already in a RAM location (location DiffK) as 1/2h and not as "h" because it is more efficient to multiply than divide.

After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, differentiation may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Diff" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

TABLE 6: DOUBLE PRECISION SQUARE ROOT

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Sqrt	22	3300 (approx.)	6	used

TABLE 7: BCD ROUTINES

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
BCDtoB	BCD to Binary	30	112	0	used
B2_BCD_Looped	Binary to BCD (16 bit) looped code	32	750	1	used
B2_BCD_Straight	Binary to BCD (16 bit) straight line code	44	572	1	used
BinBCD	Binary to BCD (8 bit)	10	62	0	unused
BCDAdd	BCD addition	5	5	0	used

© 1994 Microchip Technology Incorporated

NUMERICAL INTEGRATION

This routine performs numerical integration using Simpson's Three-Eighths Rule. This is a third order approximation for the function, whose integral is to be computed at a given point. Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. Given a function f(t), its integral over a range t_0 to t_3 is represented as:

t3

$\int$f(t)dt. This function is approximated as follows:

t0

Simpson's Three-Eighths Rule:

t3

$$\int f(t)dt = \frac{3h}{8}\,[f(t_0) + 3f(t1) + 3f(t2) + f(t3)]$$

(t_0)

The constant 3h/8 can be computed before hand and stored in a RAM location (in location IntgKLo and IntgKHi as a 16-bit number). After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, integration may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Integrate" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

PSEUDO RANDOM NUMBER GENERATOR

This routine (subroutine "Random 16" provided in ARITH.ASM) generates a pseudo random number sequence. The random points are generated using a 16 bit register and left shifting the contents with the LSB set as shown by the following schematic.

As a test, the random points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC16/17 series). The autocorelation of the captured data is computed using a stand alone program and is shown in Figure 2. From this figure, it can be seen that the data has a strong autocorrelation only at the origin and sharply approaches to zero within a few points. This demonstrates the randomness of the data captured.

FIGURE 1

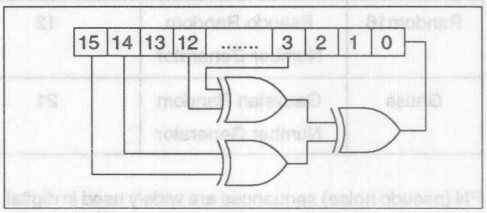


TABLE 8: DIFFERENTIATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Diff	Numerical Differentiation	34	365	10	used

TABLE 9: INTEGRATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Integrate	Numerical Integration	39	370	12	used

© 1994 Microchip Technology Incorporated

FIGURE 2 - AUTOCORELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

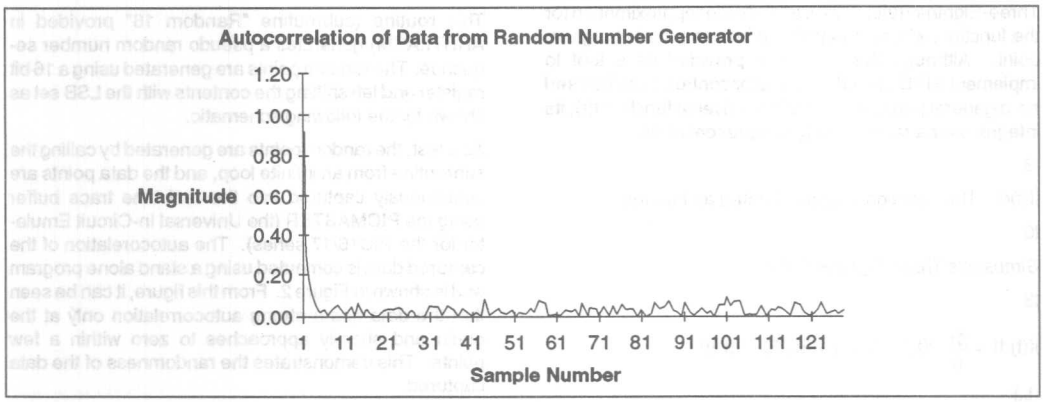
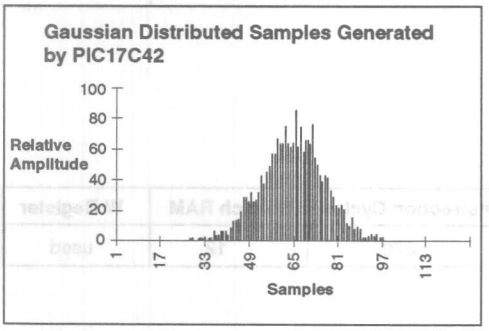


TABLE 10: RANDOM DOUBLE GENERATOR

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Random16	Pseudo Random Number Generator	12	12	0	used
Gauss	Gaussian Random Number Generator	21	452	4	used

PN (pseudo noise) sequences are widely used in digital communication systems for synchronization. These code words can also be used for data scrambling because of their good correlation properties. An interesting application of these sequences is system integrity. For example, these sequences can be regularly transmitted to a processor whose watchdog timer will time out if, say, two consecutive PN sequences do not match.

FIGURE 3 - HISTOGRAM OF THE DATA GENERATED BY THE GAUSSIAN GENERATOR



GAUSSIAN DISTRIBUTED RANDOM NUMBER GENERATOR

This routine (subroutine "Gauss" provided in ARITH.ASM) generates a sequence of random numbers with a characteristic of a normal distribution (Gaussian distributed points). This routine calls the pseudo random number generator ("random16") to obtain a near uniformly distributed random points and from these points, the Gaussian distributed points are generated. The method of generating Gaussian points is based on the "Central Limit Theorem", which states that an ensemble of average weighted sum of a sequence of uncorrelated samples tends to have a Gaussian distribution.

As a test, the Gaussian points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PIC16/17 series). A plot of the points captured is shown in Figure 3, which shows that the random points generated have the characteristics of a Gaussian distribution.

Author: Amar Palacherla
Logic Products Division

© 1994 Microchip Technology Incorporated

APPENDIX A: GENERAL PURPOSE MATH ROUTINES LISTING FILE OF ARITH.ASM

```
MPASM B0.54                                                    PAGE  1
General Purpose Math Routines For PIC17C42 : Ver 1.0
0001                 #define PAGE    EJECT
                          TITLE   "General Purpose Math Routines For PIC17C42 : Ver 1.0"
                          LIST    P=17C42, C=120, T=ON, L=0, R=DEC
                         ;
                           include "17c42.h"
                         ;
                         ;*****************************************************************
                         ;       Define RAM Locations necessary For the "ARITH.ASM"
                         ;   RAM locations should be defined before calling the library math
                         ; routines
                         ;*****************************************************************
                         ;
0001              MODE_FAST       equ     TRUE
0000              SIGNED          equ     FALSE
                         ;
                         ;*****************************************************************
                         ;
                          if MODE_FAST
                           CBLOCK  0x18
0018 0004                 ACCaLO, ACCaHI, ACCbLO, ACCbHI          ; Ram Locations for Arithmetic
001C 0004                 ACCcLO, ACCcHI, ACCdLO, ACCdHI          ;     Routines
                            ENDC
                           else
                           CBLOCK  0x20
                              ACCaLO, ACCaHI, ACCbLO, ACCbHI
                              ACCcLO, ACCcHI, ACCdLO, ACCdHI
                            ENDC
                           endif
                         ;
                           CBLOCK
0020 0004                 tempLo, tempHi, count, sign
                            ENDC
                           CBLOCK
0024 0002                 NumLo, NumHi
0026 0001                 iterCnt
                            ENDC
                         ;
                            CBLOCK                              ; RAM locations for "Diff" routine
0027 0003                       XnLo, XnHi, Xn_1_Lo
002A 0003                       Xn_1_Hi, Xn_2_Lo, Xn_2_Hi
002D 0002                       DiffKLo, DiffKHi                ; DiffK = h = Step Size
002F 0002                       DiffLo, DiffHi
                              ENDC
                         ;
                            CBLOCK                              ; RAM Locations for "Integrate"
0031 0004                       X0Lo, X0Hi, X1Lo, X1Hi          ; Routine
0035 0004                       X2Lo, X2Hi, X3Lo, X3Hi
0039 0002                       IntgKLo, IntgKHi                ; INTEGRATE CONST = 3*h/8
003B 0002                       IntgLo, IntgHi
                             ENDC
                         ;
                         ;*****************************************************************
                         ;
0018                     mulcnd  equ     ACCaLO
0019                     mulplr  equ     ACCaHI
001A                     L_byte  equ     ACCbLO
001B                     H_byte  equ     ACCbHI
                         ;
000A                     _LUPCNT equ     10              ; Set Desired Number of iterations
001E                     SqrtLo  equ     ACCdLO          ; for Square Root Routine(NEWTON Iterations)
001F                     SqrtHi  equ     ACCdHI
                         ;
                         ;  Define RAM locations for the Random Number Generators
```

```
                        ;
0018                    RandLo   equ     ACCaLO
0019                    RandHi   equ     ACCaHI          ; 16 bit Pseudo Random Number
001B                    GaussHi  equ     ACCbHI
001A                    GaussLo  equ     ACCbLO          ; 16 bit Gaussian distributed number
0020                    GaussTmp equ     tempLo
                        ;
                                ORG     0x0000
                        ;*****************************************************************
                        ;                       Math Routines Test Program
                        ;*****************************************************************
                        ;
                        ;       Load constant values to ACCa & ACCb for testing
                        ;
                        main
0000 E02D                       call    loadAB          ; result of adding ACCb+ACCa->ACCb
0001 E036                       call    D_add           ; Here Accb = 81FE
                        ;
0002 E02D                       call    loadAB          ; result of subtracting ACCb - ACCa->ACCb
0003 E03B                       call    D_sub           ; Here Accb = 7E00
                        ;
0004 E02D                       call    loadAB          ; result of multiplying ACCb*ACCa >(ACCd,ACCc)
0005 E050                       call    D_mpyS          ; Here (ACCd,ACCc) = 00FF 7E01
                        ;
0006 E02D                       call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0007 E065                       call    D_mpyF          ; Here (ACCd,ACCc) = 00FF 7E01
                        ;
0008 E02D                       call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
0009 E119                       call    D_divS          ; Here (ACCd,ACCc) = 0040 003f
                        ;
000A E02D                       call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
000B E138                       call    D_divF          ; Here (ACCd,ACCc) = 0040 003f
                        ;
000C B0F3                       movlw   0xf3
000D 0125                       movwf   NumHi
000E B0F6                       movlw   0xf6            ; Set input test number = 62454
000F 0124                       movwf   NumLo           ;  = F3F6h
0010 E27D                       call    Sqrt            ; result = 00F9h = 249 (in SqrtLo)
                        ;                               ; exact sqrt(62454) = 249.9
                        ;
0011 B0FF                       movlw   0xff
0012 0119                       movwf   mulplr          ; multiplier (in mulplr) = 0FF
0013 B0FF                       movlw   0xff            ; multiplicand(W Reg )   = 0FF
0014 0118                       movwf   mulcnd
0015 E293                       call    mpy8x8_F        ; The result 0FF*0FF = FE01 is in locations
                        ;                               ; H_byte & L_byte
0016 B0FF                       movlw   0xff
0017 0119                       movwf   mulplr          ; multiplier (in mulplr) = 0FF
0018 B0FF                       movlw   0xff            ; multiplicand(W Reg )   = 0FF
0019 0118                       movwf   mulcnd
001A E2B8                       call    mpy8x8_S        ; The result 0FF*0FF = FE01 is in
                        ;                               ; H_byte & L_byte
                        ;       Test The Random Number Generators
                        ;       Capture data into trace buffer by TABLE WRITES to a
                        ;       dummy Program Memory location
                        ;
001B B0FF                       movlw   0xff
001C 010D                       movwf   tblptrl
001D B05F                       movlw   0x5f
001E 010E                       movwf   tblptrh
                        ;
001F B030                       movlw   0x30
0020 0119                       movwf   RandHi
0021 B045                       movlw   0x45
0022 0118                       movwf   RandLo
                        ;
0023 C028                       goto    GaussPoint
                        ;
```


© 1994 Microchip Technology Incorporated

```
                        RandPoint
0024 E311                       call    Random16
0025 A418                       tlwt    _LOW,RandLo         ; only for data capture
0026 AE19                       tablwt  _HIGH,0,RandHi      ; using PICMASTER
0027 C024                       goto    RandPoint
                        ;
                        GaussPoint
0028 E31E                       call    Gauss
0029 A41A                       tlwt    _LOW,GaussLo        ; only for data capture
002A AE1B                       tablwt  _HIGH,0,GaussHi     ; using PICMASTER
002B C028                       goto    GaussPoint
                        ;
002C C02C               self    goto    self                ; End Of Test Routines
                        ;
                        loadAB
002D B001                       movlw   0x01
002E 0119                       movwf   ACCaHI
002F B0FF                       movlw   0xff                ; loads ACCa = 01FF
0030 0118                       movwf   ACCaLO
                        ;
0031 B07F                       movlw   0x7f
0032 011B                       movwf   ACCbHI
0033 B0FF                       movlw   0xFF                ; loads ACCb = 7FFF
0034 011A                       movwf   ACCbLO
0035 0002                       return
                        ;
                        ;*****************************************************************
                        ;               Double Precision Arithmetic Routines
                        ;
                        ;    Routines : Addition, Subtraction, Multiplication ,Division
                        ;               Square Root
                        ;
                        ;               NOTE :  MODE_FAST must first be set to either
                        ;                       TRUE or FALSE
                        ;
                        ;  MODE_FAST determines the RAM address locations of ACCa thru ACCd
                        ;
                        ;  If MODE_FAST is set TRUE, data transfers can be done efficiently
                        ;  using "MOVFP" & "MOVPF" instructions instead of indirectly moving
                        ;  at first to W Reg and then to the desired RAM locations
                        ;
                        ;               The speed increase using this way of locating ACCa to
                        ;         ACCd will result in a saving of about 20 Cycles/filter stage
                        ;         In this case ( a 2 stage filter), it is faster by 40 Cycles
                        ;
                        ;  If due to other constraints, ACCa thru ACCd cannot be set at
                        ; address 0x18 to 0x1f, then the user is required to set
                        ; MODE_FAST to FALSE
                        ;
                        ;*****************************************************************
                        ;               Double Precision Addition
                        ;
                        ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                        ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                        ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                        ;      (c) CALL D_add
                        ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                        ;
                        ;    Performance :
                        ;               Program Memory  :  4 (excluding call & return)
                        ;               Clock Cycles    :  4 (excluding call & return)
                        ;               W Register      :  Used
                        ;               Scratch RAM     :  0
                        ;
                        ;*****************************************************************;
                        ;
                        D_add
```

```
0036 6018                movfp    ACCaLO,wreg
0037 0F1A                addwf    ACCbLO            ;addwf lsb
0038 6019                movfp    ACCaHI,wreg
0039 111B                addwfc   ACCbHI            ;addwf msb with carry
003A 0002                return
                    ;
                    ;*****************************************************************
                    ;       ;           Double Precision Subtraction
                    ;
                    ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                    ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;       (c) CALL D_sub
                    ;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                    ;
                    ;   Performance :
                    ;                Program Memory  :   4  (excluding call & return )
                    ;                Clock Cycles    :   4  (excluding call & return )
                    ;                W Register      :   Used
                    ;                scratch RAM     :   0
                    ;*****************************************************************;
                    ;
                    D_sub
003B 6018                movfp    ACCaLO,wreg
003C 051A                subwf    ACCbLO
003D 6019                movfp    ACCaHI,wreg
003E 031B                subwfb   ACCbHI
003F 0002                return
                    ;
                    ;*****************************************************************
                    ;                Function to negate a 16 bit integer
                    ;  The two 8 bit integers are assumed to be in 2 consecutive
                    ; locations. Before calling this routine, FSR0 should be loaded with
                    ; the address of the lower byte.
                    ;       Assume that ALUSTA register is set for no autoincrement of
                    ; FSR0.
                    ;*****************************************************************
                    ;
                    negateAlt
0040 6000                movfp    indf0,wreg
0041 8D04                bcf      _fs1
0042 2D00                negw     indf0
0043 8504                bsf      _fs1
0044 6000                movfp    indf0,wreg
0045 2900                clrf     indf0
0046 0300                subwfb   indf0
0047 0002                return
                    ;
                    negate
0048 1300                comf     indf0
0049 8D04                bcf      _fs1
004A 1500                incf     indf0
004B 8504                bsf      _fs1
004C 9A04                btfsc    _z
004D 0700                decf     indf0
004E 1300                comf     indf0
004F 0002                return
                    ;
                    ;*****************************************************************
                    ;                       Double Precision Multiplication
                    ;
                    ;               ( Optimized for Code : Looped Code )
                    ;
                    ;  Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
                    ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;       (c) CALL D_mpyS
                    ;       (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
```


© 1994 Microchip Technology Incorporated

```
                        ;
                        ;       Performance :
                        ;       Program Memory  :   21 (UNSIGNED)
                        ;                           52 (SIGNED)
                        ;       Clock Cycles    :   242 (UNSIGNED :excluding CALL & RETURN)
                        ;                       :   254 (SIGNED :excluding CALL & RETURN)
                        ;       Scratch RAM     :   1 (used only if SIGNED arithmetic)
                        ;
                        ;       Note : The above timing is the worst case timing, when the
                        ;              register ACCb = FFFF. The speed may be improved if
                        ;              the register ACCb contains a number ( out of the two
                        ;              numbers ) with less number of 1s.
                        ;
                        ;               Double Precision Multiply ( 16x16 -> 32 )
                        ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                        ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                        ;*****************************************************************
                        ;
                         D_mpyS         ;results in ACCd(16 msb's) and ACCc(16
                        ;
                        if      SIGNED
                        CALL    S_SIGN
                        endif
                        ;
0050 2922                       clrf    count
0051 8422                       bsf     count,4         ; set count = 16
                        ;
                        if MODE_FAST
0052 5A20                       movpf   ACCbLO,tempLo
0053 5B21                       movpf   ACCbHI,tempHi
                        else
                                movfp   ACCbLO,wreg
                                movwf   tempLo
                                movfp   ACCbHI,wreg
                                movwf   tempHi
                        endif
0054 291F                       clrf    ACCdHI
0055 291E                       clrf    ACCdLO
                        ;
                        ;       shift right and addwf 16 times
                        ;
                        mpyLoop
0056 1921                       rrcf    tempHi
0057 1920                       rrcf    tempLo
0058 9004                       btfss   _carry
0059 C05E                       goto    NoAdd           ; LSB is 0, so no need to addwf
005A 6018                       movfp   ACCaLO,wreg
005B 0F1E                       addwf   ACCdLO          ;addwf lsb
005C 6019                       movfp   ACCaHI,wreg
005D 111F                       addwfc  ACCdHI          ;addwf msb
                        NoAdd
005E 191F                       rrcf    ACCdHI
005F 191E                       rrcf    ACCdLO
0060 191D                       rrcf    ACCcHI
0061 191C                       rrcf    ACCcLO
0062 1722                       decfsz  count
0063 C056                       goto    mpyLoop
                        ;
                        if SIGNED
                                btfss   sign,MSB
                                return
                                comf    ACCcLO
                                incf    ACCcLO
                                btfsc   _z
                                decf    ACCcHI
                                comf    ACCcHI
                                btfsc   _z
                                decf                    ACCdLO
```

```
                        comf    ACCdLO
                        btfsc   _z
                        decf    ACCdHI
                        comf    ACCdHI
                        return
                    else
0064 0002               return
                    endif
                    ;
                    ; Assemble this section only if Signed Arithmetic Needed
                    ;
                    if    SIGNED
                    ;
                    S_SIGN
                        movfp   ACCaHI,wreg
                        xorwf   ACCbHI,w
                        movwf   sign            ; MSB of sign determines whether signed
                        btfss   ACCbHI,MSB      ; if MSB set go & negate ACCb
                        goto    chek_A
                        comf    ACCbLO
                        incf    ACCbLO
                        btfsc   _z              ; negate ACCb
                        decf    ACCbHI
                        comf    ACCbHI
                    ;
                    chek_A
                        btfss   ACCaHI,MSB      ; if MSB set go & negate ACCa
                        return
                        comf    ACCaLO
                        incf    ACCaLO
                        btfsc   _z              ; negate ACCa
                        decf    ACCaHI
                        comf    ACCaHI
                        return
                    ;
                    endif
                    ;
                    ;*****************************************************************
                    ;                       Double Precision Multiplication
                    ;
                    ;               ( Optimized for Speed : straight Line Code )
                    ;
                    ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
                    ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;      (c) CALL D_mpy
                    ;      (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
                    ;
                    ;   Performance :
                    ;           Program Memory  :   179 (UNSIGNED)
                    ;                               204 (SIGNED)
                    ;           Clock Cycles    :   176 (UNSIGNED :excluding CALL & RETURN)
                    ;                           :   183 (SIGNED :excluding CALL & RETURN)
                    ;
                    ;       Note : The above timing is the worst case timing, when the
                    ;              register ACCb = FFFF. The speed may be improved if
                    ;              the register ACCb contains a number ( out of the two
                    ;              numbers ) with less number of 1s.
                    ;
                    ;              The performance specs are for Unsigned arithmetic ( i.e,
                    ;              with "SIGNED equ  FALSE ").
                    ;
                    ;              Upon return from subroutine, the input registers
```


© 1994 Microchip Technology Incorporated

```
;
;
;*************************************************************
;                   Multiplication Macro
;*************************************************************
;
mulMac  MACRO
    variable i

    i = 0

    if SIGNED
      .while i < 15
    else
      .while i < 16
    endif
      .if i < 8
        btfss    ACCbLO,i          ; test low byte
      .else
        btfss    ACCbHI,i-8        ; test high byte
      .fi
        goto     NoAdd#v(i)        ; LSB is 0, so no need to addwf
        movfp    ACCaLO,wreg
        addwf    ACCdLO            ;addwf lsb
        movfp    ACCaHI,wreg
        addwfc   ACCdHI            ;addwf msb
    NoAdd#v(i)
        rrcf     ACCdHI
        rrcf     ACCdLO
        rrcf     ACCcHI
        rrcf     ACCcLO
        bcf      _carry
        i = i+1
      .endw
    if SIGNED
        rrcf     ACCdHI
        rrcf     ACCdLO
        rrcf     ACCcHI
        rrcf     ACCcLO
        bcf      _carry
    endif
    ;
    ENDM
    ;
    ;*************************************************************
    ;                      Double Precision Negate Macros
    ;*************************************************************
    AltNegMac    MACRO            fileRegLo,fileRegHi
        movfp    fileRegLo,wreg
        negw     fileRegLo
        movfp    fileRegHi,wreg
        clrf     fileRegHi
        subwfb   fileRegHi
        ENDM

    ;
    negMac  MACRO   fileRegLo, fileRegHi
        comf     fileRegLo         ; negate FileReg ( -FileReg -> FileReg )
        incf     fileRegLo
        btfsc    _z
        decf     fileRegHi
        comf     fileRegHi
    ENDM
    ;
    NegMac32     MACRO   x3,x2,x1,x0
        movfp    x3,wreg
        negw     x3
        movfp    x2,wreg
        clrf     x2
```

© 1994 Microchip Technology Incorporated

```
                        subwfb  x2
                        movfp   x1,wreg
                        clrf    x1
                        subwfb  x1
                        movfp   x0,wreg
                        clrf    x0
                        subwfb  x0
                                ENDM
                ;
                ;*****************************************************************;
                ;               Double Precision Multiply ( 16x16 -> 32 )
                ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                ;
                D_mpyF                          ;results in ACCd(16 msb's) and ACCc(16
                ;
                if   SIGNED
                ;
                        movfp   ACCaHI,wreg
                        xorwf   ACCbHI,w
                        movwf   sign
                        btfss   ACCbHI,MSB      ; if MSB set go & negate ACCb
                        goto    chek_A_MSB_MPY
                ;
                        negMac  ACCbLO,ACCbHI
                ;
                chek_A_MSB_MPY
                        btfss   ACCaHI,MSB      ; if MSB set go & negate ACCa
                        goto    continue_MPY
                        negMac  ACCaLO,ACCaHI
                ;
                        endif
                ;
                continue_MPY
0065 291F               clrf    ACCdHI
0066 291E               clrf    ACCdLO
0067 8804               bcf     carry
                ;
                ;       use the mulMac macro 16 times
                ;
                mulMac
0000                    variable i
0000                    i = 0
                      if SIGNED
                        .while i < 15
                      else
                        .while i < 16
                endif
                        .if i < 8
                        btfss   ACCbLO,i        ; test low byte
                        .else
                        btfss   ACCbHI,i-8      ; test high byte
                        .fi
                        goto    NoAdd#v(i)      ; LSB is 0, so no need to addwf
                        movfp   ACCaLO,wreg
                        addwf   ACCdLO          ;addwf lsb
                        movfp   ACCaHI,wreg
                        addwfc  ACCdHI          ;addwf msb
                NoAdd#v(i)
                        rrcf    ACCdHI
                        rrcf    ACCdLO
                        rrcf    ACCcHI
                        rrcf    ACCcLO
                        bcf     _carry
                        i = i+1
                .endw
                        .if i < 8
0068 901A               btfss   ACCbLO,i        ; test low byte
                        .else
```


© 1994 Microchip Technology Incorporated

```
                                    btfss       ACCbHI,i-8      ; test high byte

                                  .fi

0069 C06E                         goto    NoAdd0              ; LSB is 0, so no need to addwf
006A 6018                         movfp    ACCaLO,wreg
006B 0F1E                         addwf     ACCdLO           ;addwf lsb
006C 6019                         movfp    ACCaHI,wreg
006D 111F                         addwfc    ACCdHI          ;addwf msb
                    NoAdd0
006E 191F                         rrcf      ACCdHI
006F 191E                         rrcf      ACCdLO
0070 191D                         rrcf      ACCcHI
0071 191C                         rrcf      ACCcLO
0072 8804                         bcf       _carry
0001                              i = i+1

                                  .if i < 8
0073 911A                         btfss      ACCbLO,i         ; test low byte
                                  .else
                                     btfss      ACCbHI,i-8      ; test high byte

                                  .fi

0074 C079                         goto    NoAdd1              ; LSB is 0, so no need to addwf
0075 6018                         movfp    ACCaLO,wreg
0076 0F1E                         addwf     ACCdLO           ;addwf lsb
0077 6019                         movfp    ACCaHI,wreg
0078 111F                         addwfc    ACCdHI          ;addwf msb
                    NoAdd1
0079 191F                         rrcf      ACCdHI
007A 191E                         rrcf      ACCdLO
007B 191D                         rrcf      ACCcHI
007C 191C                         rrcf      ACCcLO
007D 8804                         bcf       _carry
0002                              i = i+1

                                  .if i < 8
007E 921A                         btfss      ACCbLO,i         ; test low byte
                                  .else
                                     btfss      ACCbHI,i-8      ; test high byte

                                  .fi

007F C084                         goto    NoAdd2              ; LSB is 0, so no need to addwf
0080 6018                         movfp    ACCaLO,wreg
0081 0F1E                         addwf     ACCdLO           ;addwf lsb
0082 6019                         movfp    ACCaHI,wreg
0083 111F                         addwfc    ACCdHI          ;addwf msb
                    NoAdd2
0084 191F                         rrcf      ACCdHI
0085 191E                         rrcf      ACCdLO
0086 191D                         rrcf      ACCcHI
0087 191C                         rrcf      ACCcLO
0088 8804                         bcf       _carry
0003                              i = i+1

                                  .if i < 8
0089 931A                         btfss      ACCbLO,i         ; test low byte
                                  .else
                                     btfss      ACCbHI,i-8      ; test high byte

                                  .fi

008A C08F                         goto    NoAdd3              ; LSB is 0, so no need to addwf
008B 6018                         movfp    ACCaLO,wreg
008C 0F1E                         addwf     ACCdLO           ;addwf lsb
008D 6019                         movfp    ACCaHI,wreg
008E 111F                         addwfc    ACCdHI          ;addwf msb
```

© 1994 Microchip Technology Incorporated

```
                        NoAdd3
008F 191F                   rrcf    ACCdHI
0090 191E                   rrcf    ACCdLO
0091 191D                   rrcf    ACCcHI
0092 191C                   rrcf    ACCcLO
0093 8804                   bcf     _carry
0004                        i = i+1
                        .if i < 8
0094 941A                   btfss   ACCbLO,i        ; test low byte
                        .else
                            btfss   ACCbHI,i-8      ; test high byte
                        .fi
0095 C09A                    goto   NoAdd4          ; LSB is 0, so no need to addwf
0096 6018                    movfp  ACCaLO,wreg
0097 0F1E                    addwf  ACCdLO          ;addwf lsb
0098 6019                    movfp  ACCaHI,wreg
0099 111F                    addwfc ACCdHI          ;addwf msb
                        NoAdd4
009A 191F                   rrcf    ACCdHI
009B 191E                   rrcf    ACCdLO
009C 191D                   rrcf    ACCcHI
009D 191C                   rrcf    ACCcLO
009E 8804                   bcf     _carry
0005                        i = i+1
                        .if i < 8
009F 951A                   btfss   ACCbLO,i        ; test low byte
                        .else
                            btfss   ACCbHI,i-8      ; test high byte
                        .fi
00A0 C0A5                    goto   NoAdd5          ; LSB is 0, so no need to addwf
00A1 6018                    movfp  ACCaLO,wreg
00A2 0F1E                    addwf  ACCdLO          ;addwf lsb
00A3 6019                    movfp  ACCaHI,wreg
00A4 111F                    addwfc ACCdHI          ;addwf msb
                        NoAdd5
00A5 191F                   rrcf    ACCdHI
00A6 191E                   rrcf    ACCdLO
00A7 191D                   rrcf    ACCcHI
00A8 191C                   rrcf    ACCcLO
00A9 8804                   bcf     _carry
0006                        i = i+1
                        .if i < 8
00AA 961A                   btfss   ACCbLO,i        ; test low byte
                        .else
                            btfss   ACCbHI,i-8      ; test high byte
                        .fi
00AB C0B0                    goto   NoAdd6          ; LSB is 0, so no need to addwf
00AC 6018                    movfp  ACCaLO,wreg
00AD 0F1E                    addwf  ACCdLO          ;addwf lsb
00AE 6019                    movfp  ACCaHI,wreg
00AF 111F                    addwfc ACCdHI          ;addwf msb
                        NoAdd6
00B0 191F                   rrcf    ACCdHI
00B1 191E                   rrcf    ACCdLO
00B2 191D                   rrcf    ACCcHI
00B3 191C                   rrcf    ACCcLO
00B4 8804                   bcf     _carry
0007                        i = i+1
                        .if i < 8
00B5 971A                   btfss   ACCbLO,i        ; test low byte
                        .else
                            btfss   ACCbHI,i-8      ; test high byte
                        .fi

00B6 C0BB                    goto   NoAdd7          ; LSB is 0, so no need to addwf
00B7 6018                    movfp  ACCaLO,wreg
00B8 0F1E                    addwf  ACCdLO          ;addwf lsb
00B9 6019                    movfp  ACCaHI,wreg
00BA 111F                    addwfc ACCdHI          ;addwf msb
```


© 1994 Microchip Technology Incorporated

```
                        NoAdd7
00BB 191F                   rrcf    ACCdHI
00BC 191E                   rrcf    ACCdLO
00BD 191D                   rrcf    ACCcHI
00BE 191C                   rrcf    ACCcLO
00BF 8804                   bcf     _carry
0008                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00C0 901B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00C1 C0C6                   goto    NoAdd8          ; LSB is 0, so no need to addwf
00C2 6018                   movfp   ACCaLO,wreg
00C3 0F1E                   addwf   ACCdLO          ;addwf lsb
00C4 6019                   movfp   ACCaHI,wreg
00C5 111F                   addwfc  ACCdHI          ;addwf msb
                        NoAdd8
00C6 191F                   rrcf    ACCdHI
00C7 191E                   rrcf    ACCdLO
00C8 191D                   rrcf    ACCcHI
00C9 191C                   rrcf    ACCcLO
00CA 8804                   bcf     _carry
0009                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00CB 911B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00CC C0D1                   goto    NoAdd9          ; LSB is 0, so no need to addwf
00CD 6018                   movfp   ACCaLO,wreg
00CE 0F1E                   addwf   ACCdLO          ; addwf lsb
00CF 6019                    movfp  ACCaHI,wreg
00D0 111F                   addwfc  ACCdHI          ; addwf msb
                        NoAdd9
00D1 191F                   rrcf    ACCdHI
00D2 191E                   rrcf    ACCdLO
00D3 191D                   rrcf    ACCcHI
00D4 191C                   rrcf    ACCcLO
00D5 8804                   bcf      _carry
000A                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00D6 921B                    btfss  ACCbHI,i-8      ; test high byte
                        .fi
00D7 C0DC                   goto    NoAdd10         ; LSB is 0, so no need to addwf
00D8 6018                   movfp   ACCaLO,wreg
00D9 0F1E                   addwf   ACCdLO          ; addwf lsb
00DA 6019                   movfp   ACCaHI,wreg
00DB 111F                   addwfc  ACCdHI          ; addwf msb
                        NoAdd10
00DC 191F                   rrcf    ACCdHI
00DD 191E                   rrcf    ACCdLO
00DE 191D                   rrcf    ACCcHI
00DF 191C                   rrcf    ACCcLO
00E0 8804                   bcf     _carry
000B                        i = i+1
                        .if i < 8
                            btfss   ACCbLO,i        ; test low byte
                        .else
00E1 931B                   btfss   ACCbHI,i-8      ; test high byte
                        .fi
00E2 C0E7                   goto    NoAdd11         ; LSB is 0, so no need to addwf
00E3 6018                   movfp   ACCaLO,wreg
00E4 0F1E                   addwf   ACCdLO          ;addwf lsb
00E5 6019                   movfp   ACCaHI,wreg
```


© 1994 Microchip Technology Incorporated

```
00E6 111F                       addwfc  ACCdHI          ;addwf msb
                              NoAdd11
00E7 191F                       rrcf    ACCdHI
00E8 191E                       rrcf    ACCdLO
00E9 191D                       rrcf    ACCcHI
00EA 191C                       rrcf    ACCcLO
00EB 8804                       bcf     _carry
000C                            i = i+1
                              .if i < 8
                                btfss   ACCbLO,i        ; test low byte
                              .else
00EC 941B                       btfss   ACCbHI,i-8      ; test high byte
                              .fi
00ED C0F2                       goto    NoAdd12         ; LSB is 0, so no need to addwf
00EE 6018                       movfp   ACCaLO,wreg
00EF 0F1E                       addwf   ACCdLO          ;addwf lsb
00F0 6019                       movfp   ACCaHI,wreg
00F1 111F                       addwfc  ACCdHI          ;addwf msb
                              NoAdd12
00F2 191F                       rrcf    ACCdHI
00F3 191E                       rrcf    ACCdLO
00F4 191D                       rrcf    ACCcHI
00F5 191C                       rrcf    ACCcLO
00F6 8804                       bcf     _carry
000D                            i = i+1
                              .if i < 8
                                btfss   ACCbLO,i        ; test low byte
                              .else
00F7 951B                       btfss   ACCbHI,i-8      ; test high byte
                              .fi
00F8 C0FD                       goto    NoAdd13         ; LSB is 0, so no need to addwf
00F9 6018                       movfp   ACCaLO,wreg
00FA 0F1E                       addwf   ACCdLO          ;addwf lsb
00FB 6019                       movfp   ACCaHI,wreg
00FC 111F                       addwfc  ACCdHI          ;addwf msb
                              NoAdd13
00FD 191F                       rrcf    ACCdHI
00FE 191E                       rrcf    ACCdLO
00FF 191D                       rrcf    ACCcHI
0100 191C                       rrcf    ACCcLO
0101 8804                       i = i+1
                              .if i < 8
                                btfss   ACCbLO,i        ; test low byte
                              .else
0102 961B                       btfss   ACCbHI,i-8      ; test high byte
                              .fi
0103 C108                       goto    NoAdd14         ; LSB is 0, so no need to addwf
0104 6018                       movfp   ACCaLO,wreg
0105 0F1E                       addwf   ACCdLO          ;addwf lsb
0106 6019                       movfp   ACCaHI,wreg
0107 111F                       addwfc  ACCdHI          ;addwf msb
                              NoAdd14
0108 191F                       rrcf    ACCdHI
0109 191E                       rrcf    ACCdLO
010A 191D                       rrcf    ACCcHI
010B 191C                       rrcf    ACCcLO
010C 8804                       bcf     _carry
000F                            i = i+1
                              .if i < 8
                                btfss   ACCbLO,i        ; test low byte
                              .else
010D 971B                       btfss   ACCbHI,i-8      ; test high byte
                              .fi
010E C113                       goto    NoAdd15         ; LSB is 0, so no need to addwf
010F 6018                       movfp   ACCaLO,wreg
0110 0F1E                       addwf   ACCdLO          ;addwf lsb
0111 6019                       movfp   ACCaHI,wreg
```


© 1994 Microchip Technology Incorporated

```
0112 111F                addwfc   ACCdHI              ;addwf msb
                        NoAdd15
0113 191F                rrcf     ACCdHI
0114 191E                rrcf     ACCdLO
0115 191D                rrcf     ACCcHI
0116 191C                rrcf     ACCcLO
0117 8804                bcf      _carry
0010                     i = i+1
                        if SIGNED
                            rrcf     ACCdHI
                            rrcf     ACCdLO
                            rrcf     ACCcHI
                            rrcf     ACCcLO
                            bcf      _carry
                        endif
                    ;
                    ;
                        if SIGNED
                            btfss    sign,MSB          ; negate (ACCc,ACCd)
                            return
                    NegMac32 ACCcHI,ACCcLO,ACCdHI, ACCdLO
                            return
                            else
0118 0002                   return
                    endif
                    ;
                    ;*****************************************************************
                    ;                       Double Precision Division
                    ;
                    ;               ( Optimized for Code : Looped Code )
                    ;
                    ;*****************************************************************;
                    ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
                    ;                                               Remainder in ACCc (16 bits)
                    ;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
                    ;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
                    ;       (c) CALL D_div
                    ;       (d) The 16 bit result is in location ACCbHI & ACCbLO
                    ;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
                    ;
                    ;   Performance :
                    ;            Program Memory  :  31 (UNSIGNED)
                    ;                               39 (SIGNED)
                    ;               Clock Cycles    : 300 (UNSIGNED : excluding CALL & RETURN)
                    ;                               : 312 (SIGNED : excluding CALL & RETURN)
                    ;
                    ;       NOTE :
                    ;            The performance specs are for Unsigned arithmetic ( i.e,
                    ;               with "SIGNED equ  FALSE ").
                    ;
                    ;*****************************************************************
                    ;       Double Precision Divide ( 16/16 -> 16 )
                    ;
                    ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ):16 bit output
                    ; with Quotiont in ACCb(ACCbHI,ACCbLO)and Remainder in ACCc (ACCcHI,ACCcLO).
                    ;
                    ;               B/A = (Q) + (R)/A
                    ;       or      B = A*Q + R
                    ;
                    ;                       where   B :     Numerator
                    ;                               A :     Denominator
                    ;                               Q :     Quotiont (Integer Result)
                    ;                               R :     Remainder
                    ;
                    ;      Note :  Check for ZERO Denominator or Numerator is not performed
                    ;               A ZERO Denominator will produce incorrect results
                    ;
                    ;  SIGNED Arithmetic :
                    ;  In case of signed arithmetic, if either
```

```
                        ;  numerator or denominator is negative, then both Q & R are
                        ;  represented as negative numbers
                        ;                -(B/A) = -(Q) + (-R)/A
                        ;       or       -B = (-Q)*A + (-R)
                        ;
                        ;*****************************************************************
                        ;
                        D_divS
                        ;
0119 8404                   bsf     _fs0
011A 8504                   bsf     _fs1            ; set no auto-incrment for fsr0

                        if    SIGNED
                           CALL     S_SIGN
                        endif
                        ;
011B 2922                   clrf    count
011C 8422                   bsf     count,4         ; set count = 16
011D 291D                   clrf    ACCcHI
011E 291C                   clrf    ACCcLO
011F 291E                   clrf    ACCdLO
0120 291F                   clrf    ACCdHI
                        ;
                        ;  Looped code
                        ;
                        divLoop
0121 8804                   bcf     _carry
0122 1B1A                   rlcf    ACCbLO
0123 1B1B                   rlcf    ACCbHI
0124 1B1C                   rlcf    ACCcLO
0125 1B1D                   rlcf    ACCcHI
0126 6019                   movfp   ACCaHI,wreg
0127 041D                   subwf   ACCcHI,w        ;check if a>c
0128 9204                   btfss   _z
0129 C12C                   goto    notz
012A 6018                   movfp   ACCaLO,wreg
012B 041C                   subwf   ACCcLO,w        ; if msb equal then check lsb
                        notz
012C 9004                   btfss   _carry          ; carry set if c>a
012D C133                   goto    nosub           ; if c < a
                        subca
012E 6018                   movfp   ACCaLO,wreg     ; c-a into c
012F 051C                   subwf   ACCcLO
0130 6019                   movfp   ACCaHI,wreg
0131 031D                   subwfb  ACCcHI
0132 8004                   bsf     _carry          ;shift a 1 into d (result)
                        nosub
0133 1B1E                   rlcf    ACCdLO
0134 1B1F                   rlcf    ACCdHI
0135 1722                   decfsz  count
0136 C121                   goto    divLoop

                        ;
                        if SIGNED
                           btfss    sign,MSB
                           return
                           movlw    ACCcLO
                           movwf    fsr0
                           call     negate
                           movlw    ACCdLO
                           movwf    fsr0
                           call     negate
                           return
                           else
0137 0002                  return
                        endif
                        ;
                        ;*****************************************************************
                        ;                       Double Precision Division
```


© 1994 Microchip Technology Incorporated

```
;
;               ( Optimized for Speed : straight Line Code )
;
;*****************************************************************;
;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;                                          Remainder in ACCc (16 bits)
;
;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
;       (c) CALL D_div
;       (d) The 16 bit result is in location ACCbHI & ACCbLO
;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;
;               B/A = (Q) + (R)/A
;       or      B = A*Q + R
;
;                       where   B :     Numerator
;                               A :     Denominator
;                               Q :     Quotient (Integer Result)
;                               R :     Remainder
;
;       Note :  Check for ZERO Denominator or Numerator is not performed
;               A ZERO Denominator will produce incorrect results
;
;       SIGNED Arithmetic :
;                       In case of signed arithmetic, if either
;  numerator or denominator is negative, then both Q & R are
;  represented as negative numbers
;               -(B/A) = -(Q) + (-R)/A
;       or      -B = (-Q)*A + (-R)
;
;   Performance :
;           Program Memory      : 325 (UNSIGNED)
;                                 354 (SIGNED)
;               Clock Cycles    : 250 (UNSIGNED : excluding CALL & RETURN)
;                               : 260 (SIGNED : excluding CALL & RETURN)
;
;*****************************************************************;
;       division macro
;
divMac  MACRO
  variable i

      i = 0
     .while i < 16
;
            bcf       _carry
            rlcf      ACCbLO
            rlcf      ACCbHI
            rlcf      ACCcLO
            rlcf      ACCcHI
            movfp     ACCaHI,wreg
            subwf     ACCcHI,w          ;check if a>c
            btfss     _z
            goto      notz#v(i)
            movfp     ACCaLO,wreg
            subwf     ACCcLO,w          ;if msb equal then check lsb
notz#v(i)   btfss     _carry            ;carry set if c>a
            goto      nosub#v(i)        ; if c < a
subca#v(i)  movfp     ACCaLO,wreg       ;c-a into c
            subwf     ACCcLO
            movfp     ACCaHI,wreg
            subwfb    ACCcHI
            bsf       _carry            ;shift a 1 into d (result)
nosub#v(i)  rlcf      ACCdLO
            rlcf      ACCdHI
            i=i+1
            .endw
;
```

```
                        ENDM
                        ;
                        ;*****************************************************************
                        ;       Double Precision Divide ( 16/16 -> 16 )
                        ;
                        ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                        ;           with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
                        ;           ACCcHI,ACCcLO).
                        ;
                        ;    NOTE:  Before calling this routine, the user should make sure that
                        ;           the Numerator(ACCb) is greater than Denominator(ACCa). If
                        ;           the case is not true, the user should scale either Numerator
                        ;           or Denominator or both such that Numerator is greater than
                        ;           the Denominator.
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        D_divF
                        ;
                           if   SIGNED
                            movfp    ACCaHI,wreg
                            xorwf    ACCbHI,w
                            movwf    sign
                            btfss    ACCbHI,MSB          ; if MSB set go & negate ACCb
                            goto     chek_A_MSB_DIV
                        ;
                            negMac   ACCbLO,ACCbHI
                        ;
                        chek_A_MSB_DIV
                            btfss    ACCaHI,MSB          ; if MSB set go & negate ACCa
                            goto     continue_DIV
                            negMac   ACCaLO,ACCaHI
                        endif
                        ;
                        continue_DIV
0138 291D                   clrf     ACCcHI
0139 291C                   clrf     ACCcLO
013A 291E                   clrf     ACCdLO
013B 291F                   clrf     ACCdHI
                        ;
                        ; straight line code : using the macro divMac
                        ;
                            divMac
0000                        variable i

0000                        i = 0
                              .while i < 16
                        ;
                            bcf      _carry
                            rlcf     ACCbLO
                            rlcf     ACCbHI
                            rlcf     ACCcLO
                            rlcf     ACCcHI
                            movfp    ACCaHI,wreg
                            subwf    ACCcHI,w            ;check if a>c
                            btfss    _z
                            goto     notz#v(i)
                            movfp    ACCaLO,wreg
                            subwf    ACCcLO,w            ;if msb equal then check lsb
                        notz#v(i)   btfss    _carry      ;carry set if c>a
                            goto     nosub#v(i)          ; if c < a
                         subca#v(i) movfp     ACCaLO,wreg   ;c-a into c
                            subwf    ACCcLO
                            movfp    ACCaHI,wreg
                            subwfb   ACCcHI
                            bsf      _carry              ;shift a 1 into d (result)
                         nosub#v(i) rlcf      ACCdLO
                                    rlcf      ACCdHI
```


© 1994 Microchip Technology Incorporated

```
                        i=i+1
                       .endw

                    ;
013C 8804                       bcf     _carry
013D 1B1A                       rlcf    ACCbLO
013E 1B1B                       rlcf    ACCbHI
013F 1B1C                       rlcf    ACCcLO
0140 1B1D                       rlcf    ACCcHI
0141 6019                       movfp   ACCaHI,wreg
0142 041D                       subwf   ACCcHI,w    ;check if a>c
0143 9204                       btfss   _z
0144 C147                       goto    notz0
0145 6018                       movfp   ACCaLO,wreg
0146 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
0147 9004               notz0   btfss   _carry          ;carry set if c>a
0148 C14E                       goto    nosub0          ; if c < a
0149 6018               subca0  movfp   ACCaLO,wreg     ;c-a into c
014A 051C                       subwf   ACCcLO
014B 6019                       movfp   ACCaHI,wreg
014C 031D                       subwfb  ACCcHI
014D 8004                       bsf     _carry          ;shift a 1 into d (result)
014E 1B1E               nosub0  rlcf    ACCdLO
014F 1B1F                       rlcf    ACCdHI
0001                    i = i+1
                        ;
0150 8804                       bcf     _carry
0151 1B1A                       rlcf    ACCbLO
0152 1B1B                       rlcf    ACCbHI
0153 1B1C                       rlcf    ACCcLO
0154 1B1D                       rlcf    ACCcHI
0155 6019                       movfp   ACCaHI,wreg
0156 041D                       subwf   ACCcHI,w        ;check if a>c
0157 9204                       btfss   _z
0158 C15B                       goto    notz1
0159 6018                       movfp   ACCaLO,wreg
015A 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
015B 9004               notz1   btfss   _carry          ;carry set if c>a
015C C162                       goto    nosub1          ; if c < a
015D 6018               subca1  movfp   ACCaLO,wreg     ;c-a into c
015E 051C                       subwf   ACCcLO
015F 6019                       movfp   ACCaHI,wreg
0160 031D                       subwfb  ACCcHI
0161 8004                       bsf     _carry          ;shift a 1 into d (result)
0162 1B1E               nosub1  rlcf    ACCdLO
0163 1B1F                       rlcf    ACCdHI
0002                    i = i+1
                        ;
0164 8804                       bcf     _carry
0165 1B1A                       rlcf    ACCbLO
0166 1B1B                       rlcf    ACCbHI
0167 1B1C                       rlcf    ACCcLO
0168 1B1D                       rlcf    ACCcHI
0169 6019                       movfp   ACCaHI,wreg
016A 041D                       subwf   ACCcHI,w        ;check if a>c
016B 9204                       btfss   _z
016C C16F                       goto    notz2
016D 6018                       movfp   ACCaLO,wreg
016E 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
016F 9004               notz2   btfss   _carry          ;carry set if c>a
0170 C176                       goto    nosub2          ; if c < a
0171 6018               subca2  movfp   ACCaLO,wreg     ;c-a into c
0172 051C                       subwf   ACCcLO
0173 6019                       movfp   ACCaHI,wreg
0174 031D                       subwfb  ACCcHI
0175 8004                       bsf     _carry          ;shift a 1 into d (result)
0176 1B1E               nosub2  rlcf    ACCdLO
0177 1B1F                       rlcf    ACCdHI
0003                    i = i+1
```


© 1994 Microchip Technology Incorporated

```
                        ;
0178 8804                       bcf     _carry
0179 1B1A                       rlcf    ACCbLO
017A 1B1B                       rlcf    ACCbHI
017B 1B1C                       rlcf    ACCcLO
017C 1B1D                       rlcf    ACCcHI
017D 6019                       movfp   ACCaHI,wreg
017E 041D                       subwf   ACCcHI,w          ;check if a>c
017F 9204                       btfss   _z
0180 C183                       goto    notz3
0181 6018                       movfp   ACCaLO,wreg
0182 041C                       subwf   ACCcLO,w          ;if msb equal then check lsb
0183 9004               notz3   btfss    _carry           ;carry set if c>a
0184 C18A                       goto    nosub3           ; if c < a
0185 6018               subca3  movfp   ACCaLO,wreg      ;c-a into c
0186 051C                       subwf   ACCcLO
0187 6019                       movfp   ACCaHI,wreg
0188 031D                       subwfb  ACCcHI
0189 8004                       bsf     _carry           ;shift a 1 into d (result)
018A 1B1E               nosub3  rlcf    ACCdLO
018B 1B1F                       rlcf    ACCdHI
0004                    i = i+1
                        ;
018C 8804                       bcf     _carry
018D 1B1A                       rlcf    ACCbLO
018E 1B1B                       rlcf    ACCbHI
018F 1B1C                       rlcf    ACCcLO
0190 1B1D                       rlcf    ACCcHI
0191 6019                       movfp   ACCaHI,wreg
0192 041D                       subwf   ACCcHI,w          ;check if a>c
0193 9204                       btfss   _z
0194 C197                       goto    notz4
0195 6018                       movfp   ACCaLO,wreg
0196 041C                       subwf   ACCcLO,w          ;if msb equal then check lsb
0197 9004               notz4   btfss   _carry            ;carry set if c>a
0198 C19E                       goto    nosub4           ; if c < a
0199 6018               subca4  movfp   ACCaLO,wreg       ;c-a into c
019A 051C                       subwf   ACCcLO
019B 6019                       movfp   ACCaHI,wreg
019C 031D                       subwfb  ACCcHI
019D 8004                       bsf     _carry           ;shift a 1 into d (result)
019E 1B1E               nosub4  rlcf    ACCdLO
019F 1B1F                       rlcf    ACCdHI
0005                    i = i+1
                        ;
01A0 8804                       bcf     _carry
01A1 1B1A                       rlcf    ACCbLO
01A2 1B1B                       rlcf    ACCbHI
01A3 1B1C                       rlcf    ACCcLO
01A4 1B1D                       rlcf    ACCcHI
01A5 6019                       movfp   ACCaHI,wreg
01A6 041D                       subwf   ACCcHI,w          ;check if a>c
01A7 9204                       btfss    _z
01A8 C1AB                       goto    notz5
01A9 6018                       movfp   ACCaLO,wreg
01AA 041C                       subwf   ACCcLO,w          ;if msb equal then check lsb
01AB 9004               notz5   btfss   _carry            ;carry set if c>a
01AC C1B2                       goto    nosub5           ; if c < a
01AD 6018               subca5  movfp   ACCaLO,wreg       ;c-a into c
01AE 051C                       subwf   ACCcLO
01AF 6019                       movfp   ACCaHI,wreg
01B0 031D                       subwfb  ACCcHI
01B1 8004                       bsf     _carry          ;shift a 1 into d (result)
01B2 1B1E               nosub5  rlcf    ACCdLO
01B3 1B1F                       rlcf    ACCdHI
0006                    i = i+1

                        ;
01B4 8804                       bcf     _carry
```


© 1994 Microchip Technology Incorporated

```
01B9 6019                       movfp   ACCaHI,wreg
01BA 041D                       subwf   ACCcHI,w      ;check if a>c
01BB 9204                       btfss   _z
01BC C1BF                       goto    notz6
01BD 6018                       movfp   ACCaLO,wreg
01BE 041C                       subwf   ACCcLO,w      ;if msb equal then check lsb
01BF 9004               notz6   btfss   _carry        ;carry set if c>a
01C0 C1C6                       goto    nosub6        ; if c < a
01C1 6018               subca6  movfp   ACCaLO,wreg      ;c-a into c
01C2 051C                       subwf   ACCcLO
01C3 6019                       movfp   ACCaHI,wreg
01C4 031D                       subwfb  ACCcHI
01C5 8004                       bsf     _carry        ;shift a 1 into d (result)
01C6 1B1E               nosub6  rlcf    ACCdLO
01C7 1B1F                       rlcf    ACCdHI
0007                    i = i+1
                        ;
01C8 8804                       bcf     _carry
01C9 1B1A                       rlcf    ACCbLO
01CA 1B1B                       rlcf    ACCbHI
01CB 1B1C                       rlcf    ACCcLO
01CC 1B1D                       rlcf    ACCcHI
01CD 6019                       movfp   ACCaHI,wreg
01CE 041D                       subwf   ACCcHI,w      ;check if a>c
01CF 9204                       btfss   _z
01D0 C1D3                       goto    notz7
01D1 6018                       movfp   ACCaLO,wreg
01D2 041C                       subwf   ACCcLO,w      ;if msb equal then check lsb
01D3 9004               notz7   btfss   _carry        ;carry set if c>a
01D4 C1DA                       goto    nosub7        ; if c < a
01D5 6018               subca7  movfp   ACCaLO,wreg      ;c-a into c
01D6 051C                       subwf   ACCcLO
01D7 6019                       movfp   ACCaHI,wreg
01D8 031D                       subwfb  ACCcHI
01D9 8004                       bsf     _carry        ;shift a 1 into d (result)
01DA 1B1E               nosub7  rlcf    ACCdLO
01DB 1B1F                       rlcf    ACCdHI
0008                    i = i+1

                            ;
01DC 8804                       bcf     _carry
01DD 1B1A                       rlcf    ACCbLO
01DE 1B1B                       rlcf    ACCbHI
01DF 1B1C                       rlcf    ACCcLO
01E0 1B1D                       rlcf    ACCcHI
01E1 6019                       movfp   ACCaHI,wreg
01E2 041D                       subwf   ACCcHI,w      ;check if a>c
01E3 9204                       btfss   _z
01E4 C1E7                       goto    notz8
01E5 6018                       movfp   ACCaLO,wreg
01E6 041C                       subwf   ACCcLO,w      ;if msb equal then check lsb
01E7 9004               notz8   btfss   _carry        ;carry set if c>a
01E8 C1EE                       goto    nosub8        ; if c < a
01E9 6018               subca8  movfp   ACCaLO,wreg   ;c-a into c
01EA 051C                       subwf   ACCcLO
01EB 6019                       movfp   ACCaHI,wreg
01EC 031D                       subwfb  ACCcHI
01ED 8004                       bsf     _carry        ;shift a 1 into d (result)
01EE 1B1E               nosub8  rlcf    ACCdLO
01EF 1B1F                       rlcf    ACCdHI
0009                    i = i+1

                            ;
01F0 8804                       bcf     _carry
01F1 1B1A                       rlcf    ACCbLO
```


© 1994 Microchip Technology Incorporated

```
01F2 1B1B                       rlcf    ACCbHI
01F3 1B1C                       rlcf    ACCcLO
01F4 1B1D                       rlcf    ACCcHI
01F5 6019                       movfp   ACCaHI,wreg
01F6 041D                       subwf   ACCcHI,w        ;check if a>c
01F7 9204                       btfss   _z
01F8 C1FB                       goto    notz9
01F9 6018                       movfp   ACCaLO,wreg
01FA 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
01FB 9004               notz9   btfss   _carry          ;carry set if c>a
01FC C202                       goto    nosub9          ; if c < a
01FD 6018               subca9  movfp   ACCaLO,wreg     ;c-a into c
01FE 051C                       subwf   ACCcLO
01FF 6019                       movfp   ACCaHI,wreg
0200 031D                       subwfb  ACCcHI
0201 8004                       bsf     _carry          ;shift a 1 into d (result)
0202 1B1E               nosub9  rlcf    ACCdLO
0203 1B1F                       rlcf    ACCdHI
000A                    i = i+1
                        ;
0204 8804                       bcf     _carry
0205 1B1A                       rlcf    ACCbLO
0206 1B1B                       rlcf    ACCbHI
0207 1B1C                       rlcf    ACCcLO
0208 1B1D                       rlcf    ACCcHI
0209 6019                       movfp   ACCaHI,wreg
020A 041D                       subwf   ACCcHI,w        ;check if a>c
020B 9204                       btfss   _z
020C C20F                       goto    notz10
020D 6018                       movfp   ACCaLO,wreg
020E 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
020F 9004               notz10  btfss   _carry          ;carry set if c>a
0210 C216                       goto    nosub10         ; if c < a
0211 6018               subca10 movfp   ACCaLO,wreg       ;c-a into c
0212 051C                       subwf   ACCcLO
0213 6019                       movfp   ACCaHI,wreg
0214 031D                       subwfb  ACCcHI
0215 8004                       bsf     _carry          ;shift a 1 into d (result)
0216 1B1E               nosub10 rlcf    ACCdLO
0217 1B1F                       rlcf    ACCdHI
000B                    i = i+1
                        ;
0218 8804                       bcf     _carry
0219 1B1A                       rlcf    ACCbLO
021A 1B1B                       rlcf    ACCbHI
021B 1B1C                       rlcf    ACCcLO
021C 1B1D                       rlcf    ACCcHI
021D 6019                       movfp   ACCaHI,wreg
021E 041D                       subwf   ACCcHI,w        ;check if a>c
021F 9204                       btfss   _z
0220 C223                       goto    notz11
0221 6018                       movfp   ACCaLO,wreg
0222 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
0223 9004               notz11  btfss   _carry          ;carry set if c>a
0224 C22A                       goto    nosub11         ; if c < a
0225 6018               subca11 movfp   ACCaLO,wreg     ;c-a into c
0226 051C                       subwf   ACCcLO
0227 6019                       movfp   ACCaHI,wreg
0228 031D                       subwfb  ACCcHI
0229 8004                       bsf     _carry          ;shift a 1 into d (result)
022A 1B1E               nosub11 rlcf    ACCdLO
022B 1B1F                       rlcf    ACCdHI
000C                    i = i+1
                        ;
022C 8804                       bcf     _carry
022D 1B1A                       rlcf    ACCbLO
022E 1B1B                       rlcf    ACCbHI
022F 1B1C                       rlcf    ACCcLO
0230 1B1D                       rlcf    ACCcHI
```


© 1994 Microchip Technology Incorporated

```
0231 6019                          movfp   ACCaHI,wreg
0232 041D                          subwf   ACCcHI,w        ;check if a>c
0233 9204                          btfss   _z
0234 C237                          goto    notz12
0235 6018                          movfp   ACCaLO,wreg
0236 041C                          subwf   ACCcLO,w        ;if msb equal then check lsb
0237 9004          notz12          btfss   _carry          ;carry set if c>a
0238 C23E                          goto    nosub12         ; if c < a
0239 6018          subca12         movfp   ACCaLO,wreg     ;c-a into c
023A 051C                          subwf   ACCcLO
023B 6019                          movfp   ACCaHI,wreg
023C 031D                          subwfb  ACCcHI
023D 8004                          bsf     _carry          ;shift a 1 into d (result)
023E 1B1E          nosub12         rlcf    ACCdLO
023F 1B1F                          rlcf    ACCdHI
000D               i = i+1

                   ;
0240 8804                          bcf     _carry
0241 1B1A                          rlcf    ACCbLO
0242 1B1B                          rlcf    ACCbHI
0243 1B1C                          rlcf    ACCcLO
0244 1B1D                          rlcf    ACCcHI
0245 6019                          movfp   ACCaHI,wreg
0246 041D                          subwf   ACCcHI,w        ;check if a>c
0247 9204                          btfss   _z
0248 C24B                          goto    notz13
0249 6018                          movfp   ACCaLO,wreg
024A 041C                          subwf   ACCcLO,w        ;if msb equal then check lsb
024B 9004          notz13          btfss   _carry          ;carry set if c>a
024C C252                          goto    nosub13         ; if c < a
024D 6018          subca13         movfp   ACCaLO,wreg     ;c-a into c
024E 051C                          subwf   ACCcLO
024F 6019                          movfp   ACCaHI,wreg
0250 031D                          subwfb  ACCcHI
0251 8004                          bsf     _carry          ;shift a 1 into d (result)
0252 1B1E          nosub13         rlcf    ACCdLO
0253 1B1F                          rlcf    ACCdHI
000E               i = i+1
                   ;
0254 8804                          bcf     _carry
0255 1B1A                          rlcf    ACCbLO
0256 1B1B                          rlcf    ACCbHI
0257 1B1C                          rlcf    ACCcLO
0258 1B1D                          rlcf    ACCcHI
0259 6019                          movfp   ACCaHI,wreg
025A 041D                          subwf   ACCcHI,w        ;check if a>c
025B 9204                          btfss   _z
025C C25F                          goto    notz14
025D 6018                          movfp   ACCaLO,wreg
025E 041C                          subwf   ACCcLO,w        ;if msb equal then check lsb
025F 9004          notz14          btfss   _carry          ;carry set if c>a
0260 C266                          goto    nosub14         ; if c < a
0261 6018          subca14         movfp   ACCaLO,wreg     ;c-a into c
0262 051C                          subwf   ACCcLO
0263 6019                          movfp   ACCaHI,wreg
0264 031D                          subwfb  ACCcHI
0265 8004                          bsf     _carry          ;shift a 1 into d (result)
0266 1B1E          nosub14         rlcf    ACCdLO
0267 1B1F                          rlcf    ACCdHI
000F               i = i+1
                     ;
0268 8804                          bcf     _carry
0269 1B1A                          rlcf    ACCbLO
026A 1B1B                          rlcf    ACCbHI
026B 1B1C                          rlcf    ACCcLO
026C 1B1D                          rlcf    ACCcHI
026D 6019                          movfp   ACCaHI,wreg
026E 041D                          subwf   ACCcHI,w        ;check if a>c
```


© 1994 Microchip Technology Incorporated

```
026F 9204                       btfss   _z
0270 C273                       goto    notz15
0271 6018                       movfp   ACCaLO,wreg
0272 041C                       subwf   ACCcLO,w        ;if msb equal then check lsb
0273 9004           notz15      btfss   _carry          ;carry set if c>a
0274 C27A                       goto    nosub15         ; if c < a
0275 6018           subca15     movfp   ACCaLO,wreg     ;c-a into c
0276 051C                       subwf   ACCcLO
0277 6019                       movfp   ACCaHI,wreg
0278 031D                       subwfb  ACCcHI
0279 8004                       bsf     _carry          ;shift a 1 into d (result)
027A 1B1E           nosub15     rlcf    ACCdLO
027B 1B1F                       rlcf    ACCdHI
0010                i = i+1
                    ;
                    ;
027C 027C                 if SIGNED
                                btfss   sign,MSB        ; negate (ACCc,ACCd)
                                return
                                negMac  ACCcLO,ACCcHI
                                negMac  ACCdLO,ACCdHI
                                return
                                else
027C 0002                       return
                                      endif
                          ;
                          ;*****************************************************************
                          ;
                          ;            Square Root By Newton Raphson Method
                          ;
                          ;   This routine computes the square root of a 16 bit number(with
                          ; low byte in NumLo & high byte in NumHi ). After loading NumLo &
                          ; NumHi with the desired number whose square root is to be computed,
                          ; branch to location Sqrt ( by "GOTO  Sqrt" ). " CALL  Sqrt" cannot
                          ; be issued because the Sqrt function makes calls to Math routines
                          ; and the stack is completely used up.
                          ;       The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                          ; The total number of iterations is set to ten. If more iterations
                          ; are desired, change "LupCnt equ .10" to the desired value. Also,
                          ; the initial guess value of the square root is given set as
                          ; input/2 ( in subroutine "init" ). The user may modify this scheme
                          ; if a better initial approximation value is known. A good initial
                          ; guess will help the algorithm converge at a faster rate and thus
                          ; less number of iterations required.
                          ;       Two utility math routines are used by this program : D_divS
                          ; and D_add. These two routines are listed as seperate routines
                          ; under double precision Division and double precision addition
                          ; respectively.
                          ;
                          ; Note : If square root of an 8 bit number is desired, it is probably
                          ; better to have a table look scheme rather than using numerical
                          ; methods.
                          ;        This method is computationally quite intensive and
                          ; slow, but very accurate and the convergence rate is high
                          ;
                          ;  Performance :
                          ;            Program Memory : 22 (excluding D_divS subroutine)
                          ;             Clock Cycles : 3000 (approximately,with 10 iterations)
                          ;
                          ;       The #of cycles depends on Number of Iterations Selected.
                          ; In a lot of cases 5 or less iterations may be sufficient
                          ;
                          ;
                          ;*****************************************************************
                          ;                  Newton-Raphson Method
                          ;*****************************************************************

                          Sqrt
027D E28B                   call    SqrtInit            ; compute initial sqrt = Num/2
```


© 1994 Microchip Technology Incorporated

```
                        nextIter
                           if MODE_FAST
027E 7A24                  movfp    NumLo,ACCbLO
027F 7B25                  movfp    NumHi,ACCbHI
                           else
                           movfp    NumLo,wreg
                           movwf    ACCbLO
                           movfp    NumHi,wreg
                           movwf    ACCbHI
                           endif
                        ;
0280 E119                  call     D_divS              ; double precision division
                                                                ; double precision addition
0281 601E                  movfp    ACCdLO,wreg         ; ACCd + ACCa -> ACCd
0282 0F18                  addwf    ACCaLO              ;addwf lsb
0283 601F                  movfp    ACCdHI,wreg
0284 1119                  addwfc   ACCaHI              ;addwf msb
                                                        ; now divide by 2
0285 8804                   bcf      _carry
0286 1919                   rrcf     ACCaHI
0287 1918                   rrcf     ACCaLO
                        ;
0288 1726                  decfsz   iterCnt
0289 C27E                  goto     nextIter
028A 0002                  return                       ; End Sqrt
                        ;
                        SqrtInit
028B B00A                   movlw    _LUPCNT
028C 0126                   movwf     iterCnt           ; set number of iterations
                         if MODE_FAST
028D 7925                  movfp    NumHi,ACCaHI
028E 7824                  movfp    NumLo,ACCaLO
                           else
                           movfp    NumHi,wreg
                           movwf    ACCaHI
                           movfp    NumLo,wreg          ; set initial guess root = NUM/2
                           movwf    ACCaLO
                           endif
028F 8804                  bcf      _carry
0290 1919                  rrcf     ACCaHI
0291 1918                  rrcf     ACCaLO              ; set initial sqrt = Num/2
0292 0002                  return
                        ;
                        ;*****************************************************************
                        ;                    8x8 Software Multiplier
                        ;                ( Fast Version : Straight Line Code )
                        ;
                        ;   The 16 bit result is stored in 2 bytes
                        ;
                        ; Before calling the subroutine " mpy ", the multiplier should
                        ; be loaded in location " mulplr ", and the multiplicand in
                        ; " mulcnd " . The 16 bit result is stored in locations
                        ; H_byte & L_byte.
                        ;
                        ;       Performance :
                        ;           Program Memory  :  36 words
                        ;           # of cycles     :  36       (excluding call & return)
                        ;           Scratch RAM     :   0 locations
                        ;           W Register      :  Used
                        ;
                        ;  This routine is optimized for speed efficiency ( straight line code
)
                        ;  For code efficiency, refer to "mult8x8S.asm" ( looped code )
                        ;*****************************************************************
                        ;     Define a macro for adding & right shifting
                        ;
                        multiply    MACRO
                           variable  i                  ;
                             i = 0
```

```
                                    addwf   H_byte
                                    rrcf    H_byte
                                    rrcf    L_byte
                                i = i+1 ;
                                .endw
                            ENDM                                ; End of macro
                        ;
                        ;
                        mpy8x8_F
0293 291B                           clrf    H_byte
0294 291A                           clrf    L_byte
0295 6018                           movfp   mulcnd,wreg         ; move the multiplicand to W reg.
0296 8804                           bcf     _carry              ; Clear the carry bit in the status Reg.
                        ;
                            multiply
                            variable  i                         ;
0000                        i = 0
0000                        .while i < 8
                                    btfsc   mulplr,i
                                    addwf   H_byte
                                    rrcf    H_byte
                                    rrcf    L_byte
                             i = i+1                            ;
                            .endw
0297 9819                           btfsc   mulplr,i
0298 0F1B                           addwf   H_byte
0299 191B                           rrcf    H_byte
029A 191A                           rrcf    L_byte
0001                          i = i+1               ;
029B 9919                           btfsc   mulplr,i
029C 0F1B                           addwf   H_byte
029D 191B                           rrcf    H_byte
029E 191A                           rrcf    L_byte
0002                          i = i+1               ;
029F 9A19                           btfsc   mulplr,i
02A0 0F1B                           addwf   H_byte
02A1 191B                           rrcf    H_byte
02A2 191A                           rrcf    L_byte
0003                          i = i+1               ;
02A3 9B19                           btfsc   mulplr,i
02A4 0F1B                           addwf   H_byte
02A5 191B                           rrcf    H_byte
02A6 191A                           rrcf    L_byte
0004                          i = i+1               ;
02A7 9C19                           btfsc   mulplr,i
02A8 0F1B                           addwf   H_byte
02A9 191B                           rrcf    H_byte
02AA 191A                           rrcf    L_byte
0005                          i = i+1               ;
02AB 9D19                           btfsc   mulplr,i
02AC 0F1B                           addwf   H_byte
02AD 191B                           rrcf    H_byte
02AE 191A                           rrcf    L_byte
0006                          i = i+1               ;
02AF 9E19                           btfsc   mulplr,i
02B0 0F1B                           addwf   H_byte
02B1 191B                           rrcf    H_byte
02B2 191A                           rrcf    L_byte
0007                          i = i+1               ;

02B3 9F19                   btfsc   mulplr,i
02B4 0F1B                   addwf   H_byte
02B5 191B                   rrcf    H_byte
02B6 191A                   rrcf    L_byte
0008                        i = i+1             ;
                        ;
02B7 02B7 0002          return
```


© 1994 Microchip Technology Incorporated

```
                        ;
                        ;*****************************************************************
                        ;                       8x8 Software Multiplier
                        ;                  ( Code Efficient : Looped Code )
                        ;
                        ;   The 16 bit result is stored in 2 bytes
                        ;
                        ; Before calling the subroutine " mpy ", the multiplier should
                        ; be loaded in location " mulplr ", and the multiplicand in
                        ; " mulcnd " . The 16 bit result is stored in locations
                        ; H_byte & L_byte.
                        ;
                        ;       Performance :
                        ;           Program Memory  :  13 words (excluding call & return)
                        ;           # of cycles     :  69       (excluding call & return)
                        ;           Scratch RAM     :  1 byte
                        ;           W Register      :  Used
                        ;
                        ;  This routine is optimized for code efficiency ( looped code )
                        ;  For time efficiency code refer to "mult8x8F.asm" (straight line code)
                        ;*****************************************************************
                        ;
                        mpy8x8_S
02B8 291B                   clrf    H_byte
02B9 291A                   clrf    L_byte
02BA 2922                   clrf    count
02BB 8322                   bsf     count,3         ; set count = 8
02BC 6018                   movfp   mulcnd,wreg
02BD 8804                   bcf     _carry          ; Clear the carry bit in the status Reg.
                        loop
02BE 9819                   btfsc   mulplr,0
02BF 0F1B                   addwf   H_byte
02C0 191B                   rrcf    H_byte
02C1 191A                   rrcf    L_byte
02C2 2119                   rrncf   mulplr
02C3 1722                   decfsz  count
02C4 C2BE                   goto    loop
                        ;
02C5 0002                   return
                        ;
                        ;*****************************************************************
                        ;                       Numerical Differenciation
                        ;
                        ;       The so called "Three-Point Formula" is implemented to
                        ; differenciate a sequence of points (uniformly sampled).
                        ;       The eqn implemented is :
                        ;            f'(Xn) = [ f(Xn - 2h) - 4*f(Xn - h) + 3*f(Xn)]*0.5/h
                        ;  where Xn is the present sample and 'h' is the step size.
                        ;
                        ;       The above formula may be rewritten as :
                        ;
                        ;            f'(Xn) = [ 0.5*f(Xn -2) - 2*f(Xn - 1) + 0.5*3*f(Xn)]*1/DiffK
                        ;  where DiffK = h = Step Size
                        ;
                        ;   This differenciation routine can be used very effectively
                        ; in the computation of the differential component part in
                        ; a PID Loop calculation in Motor Control Applications
                        ;
                        ;       Double precision arithmetic is used throught
                        ;  The present sample value is assumed to be in locations
                        ; (XnHi, XnLo). The past two values are assumed to be in locations
                        ; (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).
                        ;       The output value is located in DiffHi & DiffLo. No overflow
                        ; checking mechanism is implemented. If the values are limited
                        ; to 12 bits, then the user need not worry about overflows
                        ;
                        ;  It is user's responsibility to update the past values with the
                        ; present values before calling this routine.
                        ;  After computation, the present value Xn is not moved to Xn_1
```

4

```
                        ; because the user may want these values to be intact for other
                        ; computations ( say numerical integration)
                        ;    Also it is user's responsibility to set past 2 values
                        ; (Xn_1 & Xn_2) values to be zero on initialization.
                        ;
                        ;*****************************************************************
                        ;
                        Diff
02C6 602B                   movfp     Xn_2_Lo,wreg
02C7 0E27                   addwf     XnLo,w
02C8 011A                   movwf     ACCbLO
02C9 602C                   movfp     Xn_2_Hi,wreg
02CA 1028                   addwfc    XnHi,w
02CB 011B                   movwf     ACCbHI          ; Y = f(Xn-2) + f(Xn)
                        ;
02CC 6027                   movfp     XnLo,wreg
02CD 0F1A                   addwf     ACCbLO
02CE 6028                   movfp     XnHi,wreg
02CF 111B                   addwfc    ACCbHI
02D0 6027                   movfp     XnLo,wreg
02D1 0F1A                   addwf     ACCbLO
02D2 6028                   movfp     XnHi,wreg
02D3 111B                   addwfc    ACCbHI          ; Y = f(Xn-2) + 3*f(Xn)
                        ;
02D4 8804                   bcf       _carry
02D5 191B                   rrcf      ACCbHI
02D6 191A                   rrcf      ACCbLO           ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]

                        ;
02D7 6029                   movfp     Xn_1_Lo,wreg
02D8 051A                   subwf     ACCbLO
02D9 602A                   movfp     Xn_1_Hi,wreg
02DA 031B                   subwfb    ACCbHI
02DB 6029                   movfp     Xn_1_Lo,wreg
02DC 051A                   subwf     ACCbLO
02DD 602A                   movfp     Xn_1_Hi,wreg
02DE 031B                   subwfb    ACCbHI          ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)
                        ;
02DF 602D                   movfp     DiffKLo,wreg
02E0 0118                   movwf     ACCaLO
02E1 602E                   movfp     DiffKHi,wreg
02E2 0119                   movwf     ACCaHI
                        ;
02E3 E119                   call      D_divS
02E4 601A                   movfp     ACCbLO,wreg
02E5 012F                   movwf     DiffLo
02E6 601B                   movfp     ACCbHI,wreg
02E7 0130                   movwf     DiffHi          ; result = Y/h
                        ;
02E8 0002                   return
                        ;
                        ;*****************************************************************
                        ;
                        ;                       Numerical Integration
                        ;
                        ;    Simpson's Three-Eighths Rule is implemented
                        ;
                        ;    Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*3*h/8
                        ;
                        ; where 'h' is the step size and the integral is over the
                        ; range X0 to X3
                        ;      The above equation can be rewritten as
                        ;
                        ;    Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*IntgK
                        ;
                        ; where IntgK = 3*h/8 (in locations (IntgKHi, IntgKHi)
                        ;
                        ;  This Integration routine can be used very effectively
```


© 1994 Microchip Technology Incorporated

```
                        ; in the computation of the integral component part in
                        ; a PID Loop calculation in Motor Control Applications
                        ;
                        ;       Double precision arithmetic is used throught
                        ; The three input values over which the integral is to be computed
                        ; are assumed to be in locations (X0Lo,X0Hi), (X1Lo,X1Hi) , (X2Lo,X2Hi)
                        ; and (X3Lo,X3Hi)
                        ;       The output value is located in IntgHi & IntgLo. No overflow
                        ; checking mechanism is implemented. If the values are limited
                        ; to 12 bits, then the user need not worry about overflows
                        ;
                        ; It is user's responsibility to update the past values with the
                        ; present values before calling this routine.
                        ; After computation, the present value Xn is not moved to Xn_1
                        ; because the user may want these values to be intact for other
                        ; computations ( say numerical integration)
                        ;       Also it is user's responsibility to set past 2 values
                        ; (Xn_1 & Xn_2) values to be zero on initialization.
                        ;
                        ;
                        ;*****************************************************************
                        ;
                        Integrate
02E9 6031                   movfp     X0Lo,wreg
02EA 0E37                   addwf     X3Lo,w
02EB 011A                   movwf     ACCbLO
02EC 6032                   movfp     X0Hi,wreg
02ED 1038                   addwfc    X3Hi,w
02EE 011B                   movwf     ACCbHI          ; Intg = f(X0) + f(X3)
                        ;
02EF 6033                   movfp     X1Lo,wreg
02F0 0F1A                   addwf     ACCbLO
02F1 6034                   movfp     X1Hi,wreg
02F2 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +X1
02F3 6033                   movfp     X1Lo,wreg
02F4 0F1A                   addwf     ACCbLO
02F5 6034                   movfp     X1Hi,wreg
02F6 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +2*X1
02F7 6033                   movfp     X1Lo,wreg
02F8 0F1A                   addwf     ACCbLO
02F9 6034                   movfp     X1Hi,wreg
02FA 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1
                        ;
02FB 6035                   movfp     X2Lo,wreg
02FC 0F1A                   addwf     ACCbLO
02FD 6036                   movfp     X2Hi,wreg
02FE 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + X2
02FF 6035                   movfp     X2Lo,wreg
0300 0F1A                   addwf     ACCbLO
0301 6036                   movfp     X2Hi,wreg
0302 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + 2*X2
0303 6035                   movfp     X2Lo,wreg
0304 0F1A                   addwf     ACCbLO
0305 6036                   movfp     X2Hi,wreg
0306 111B                   addwfc    ACCbHI          ; Intg = f(X0) + f(X3) +3*X1 + 3*X2
                        ;
0307 6039                   movfp     IntgKLo,wreg
0308 0118                   movwf     ACCaLO
0309 603A                   movfp     IntgKHi,wreg
030A 0119                   movwf     ACCaHI          ;  ACCa=IntgK (prepare for multiplication)
                        ;
030B E050                    call     D_mpyS          ; make sure to set for either SIGNED or UNSIGNED
030C 601E                   movfp     ACCdLO,wreg
030D 013B                   movwf     IntgLo          ; 32 bit result in ACCd & ACCc
030E 601F                   movfp     ACCdHI,wreg
030F 013C                   movwf     IntgHi          ; upper 16 bits = result
                        ;
0310 0002                    return
                        ;
```

```
                        ;*****************************************************************
                        ;
                        ;                       Random Number Generator
                        ;
                        ; This routine generates a 16 Bit Pseudo Sequence Random Generator
                        ; It is based on Linear shift register feedback. The sequence
                        ; is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )
                        ;
                        ;    The 16 bit random number is in location RandHi(high byte)
                        ; & RandLo (low byte)
                        ;
                        ;       Before calling this routine, make sure the initial values
                        ; of RandHi & RandLo are NOT ZERO
                        ;       A good chiose of initial random number is 0x3045
                        ;*****************************************************************
                        ;
                        Random16
0311 1A19                   rlcf    RandHi,w
0312 0C19                   xorwf    RandHi,w
0313 1B00                   rlcf    wreg            ; carry bit = xorwf(Q15,14)
                        ;
0314 1D19                   swapf   RandHi
0315 1C18                   swapf   RandLo,w
0316 2300                   rlncf     wreg
0317 0C19                   xorwf    RandHi,w       ; LSB = xorwf(Q12,Q3)
0318 1D19                   swapf   RandHi
0319 B501                   andlw   0x01
031A 1B18                   rlcf    RandLo
031B 0D18                   xorwf    RandLo
031C 1B19                   rlcf    RandHi
031D 0002                   return
                        ;
                        ;*****************************************************************
                        ;                       Gaussian Noise Generator
                        ;
                        ; This routine generates a 16 Bit Gaussian distributed random
                        ; points. This routine calls the routine "Random16", which
                        ; generates a psuedo random noise sequence. Gaussian noise
                        ; is computed using the CENTRAL LIMIT THEOREM.
                        ;       The Central Limit Theorem states that the average weighted
                        ; sum of uncorelated samples tends to have a Gaussian distribution
                        ; For practical purposes, the sum could be over a sample size
                        ; of 32 Random numbers. Better results could result if a larger
                        ; sample size is desired. For faster results, a sum over 16 samples
                        ; would also be adequate ( say, for applications like Speech synthesis,
                        ; channel simulations, etc).
                        ;
                        ;       The 16 bit Gaussian distributed point is in locations
                        ; GaussHi & GaussLo
                        ;
                        ;       Before calling this routine, the initial seed of Random
                        ; number should be NON ZERO ( refer to notes on "Random16" routine
                        ;
                        ;*****************************************************************
                        ;
                        Gauss
031E 2922                   clrf    count
031F 8522                   bsf     count,5         ; set Sample size = 32
0320 291A                   clrf    GaussLo
0321 291B                   clrf    GaussHi
0322 2920                   clrf    GaussTmp
                        ;
                        NextGauss
0323 E311                   call    Random16        ; get a random value
0324 6018                   movfp   RandLo,wreg
0325 0F1A                   addwf   GaussLo
0326 6019                   movfp   RandHi,wreg
0327 111B                   addwfc  GaussHi
0328 2900                   clrf    wreg
```


© 1994 Microchip Technology Incorporated

```
0329 1120                   addwfc  GaussTmp
032A 1722                   decfsz  count
032B C323                   goto    NextGauss       ; sum 16 random numbers
                    ;
032C B005                   movlw   5
                    GaussDiv16
032D 1920                   rrcf    GaussTmp
032E 191B                   rrcf    GaussHi
032F 191A                   rrcf    GaussLo         ; weghted average
0330 1700                   decfsz  wreg            ; divide by 32
0331 C32D                   goto    GaussDiv16
                    ;
0332 0002                   return
                    ;
                            END                     ; End Of arith.asm

Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Incorporated

APPENDIX B : BCD ARITHMETIC ROUTINES LISTING FILE OF BCD.ASM

```
MPASM B0.54                                                    PAGE  1

0001                #define PAGE    EJECT
                    ;TITLE   "BCD Arithmetic Routines : Ver 1.0"
                    ;*****************************************************************
                    ;                        BCD Arithmetic Routines
                    ;*****************************************************************

                    LIST     P=17C42, C=80, L=0, R=DEC
                    include "17c42.h"
                            CBLOCK   0x20
0020 0002                   Lbyte, Hbyte
0022 0003                   R2, R1, R0                ;must maintain R2, R1, R0 sequence
0025 0001                   count
0026 0002                   Num1, Num2
                            ENDC
                    ;
0026                BCD     equ       Num1
0026                Htemp   equ       Num1
0027                Ltemp   equ       Num2
                    ;
                         ORG          0x0000
                    ;*******************************************************

                    ;                        BCD Arithmetic Test Program
                    ;*******************************************************
                    ;
                    main
0000 2B21                   setf      Hbyte
0001 2B20                   setf      Lbyte
                    ;                                     ; 16 bit binary num = 0xfff
0002 E01F                   call      B2_BCD_Looped       ; after conversion the Deci

                    ;                                     ; in R0, R1, R2 = 06,55,35
0003 2B21                   setf      Hbyte
0004 2B20                   setf      Lbyte
0005 E03F                   call      B2_BCD_Straight  ; same as above, but straig
                    ;
0006 B006                   movlw     0x06
0007 0124                   movwf     R0
0008 B055                   movlw     0x55
0009 0123                   movwf     R1
000A B035                   movlw     0x35
000B 0122                   movwf     R2                ; setf R0R1R2 = 65535
                    ;
000C E082                   call      BCDtoB            ; after conversion Hbyte =
                    ;                                     ; and Lbyte = 0xff
000D B099                   movlw     0x99
000E 0126                   movwf     Num1
000F B099                   movlw     0x99
0010 0127                   movwf     Num2              ; setf Num1 = Num2 = 0x99
                    ;
0011 E093                   call      BCDAdd            ; after addition, Num2 = 98
                    ;                                     ; and Num1 = 01 ( 99+99 = 1
                    ;
0012 B063                   movlw    0x63              ; setf Wreg = 63 hex
0013 E015                   call      BinBCD              ; after conversion, BCD = 9
                    ;                                      ; 63 hex = 99 decimal.
                    ;
0014 C014           self    goto    self
                    ;
                    ;*******************************************************
                    ;                  Binary To BCD Conversion Routine (8 bit)
```


© 1994 Microchip Technology Incorporated

```
                    ;
                    ;       This routine converts the 8 bit binary number in th
                    ; to a 2 digit BCD number in location BCD( compacted BCD Co
                    ;       The least significant digit is returned in location
                    ; the most significant digit is returned in location MSD.
                    ;
                    ;   Performance :
                    ;               Program Memory  :  10
                    ;               Clock Cycles    :  62  (worst case when W =
                    ;                                      ( i.e max Decimal nu
                    ;*****************************************************
                    ;
                        BinBCD
0015 2926               clrf    BCD
                        again
0016 B1F6               addlw   -10
0017 9004               btfss   _carry
0018 C01B               goto    swapBCD
0019 1526               incf    BCD
001A C016               goto    again
                        swapBCD
001B B10A               addlw   10
001C 1D26               swapf   BCD
001D 0926               iorwf   BCD
001E 0002               return
                    ;
                    ;*****************************************************
                    ;                   Binary To BCD Conversion Routine (16 Bit)
                    ;                           (LOOPED Version)
                    ;
                    ;       This routine converts a 16 Bit binary Number to a 5
                    ; BCD Number.
                    ;
                    ;       The 16 bit binary number is input in locations Hbyt
                    ; Lbyte with the high byte in Hbyte.
                    ;       The 5 digit BCD number is returned in R0, R1 and R2
                    ; containing the MSD in its right most nibble.
                    ;
                    ;   Performance :
                    ;               Program Memory  :  32
                    ;               Clock Cycles    :  750
                    ;*****************************************************
                    ;
                        B2_BCD_Looped
001F 8404               bsf     _fs0
0020 8504               bsf     _fs1            ; set fsr0 for no auto inc
                    ;
0021 8804               bcf     _carry
0022 2925               clrf    count
0023 8425               bsf     count,4         ; set count = 16
0024 2924               clrf    R0
0025 2923               clrf    R1
0026 2922               clrf    R2
                        loop16a
0027 1B20               rlcf    Lbyte
0028 1B21               rlcf    Hbyte
0029 1B22               rlcf    R2
002A 1B23               rlcf    R1
002B 1B24               rlcf    R0
                    ;
002C 2725               dcfsnz  count
002D 0002               return
                        adjDEC
002E B022               movlw   R2              ; load R2 as indirect addr
002F 0101               movwf   fsr0
0030 E036               call    adjBCD
                    ;
0031 1501               incf    fsr0
0032 E036               call    adjBCD
```

```
                        ;
0033 1501               incf    fsr0
0034 E036               call    adjBCD
                        ;
0035 C027               goto    loop16a
                        ;
                        adjBCD
0036 6000                 movfp   indf0,wreg
0037 B103                 addlw   0x03
0038 9B00                 btfsc   wreg,3          ; test if result > 7
0039 0100                 movwf   indf0
003A 6000                 movfp   indf0,wreg
003B B130                 addlw   0x30
003C 9F00                 btfsc   wreg,7          ; test if result > 7
003D 0100                 movwf   indf0           ; save as MSD
003E 0002                 return
                        ;
                        ;*******************************************************
                        ;               Binary To BCD Conversion Routine (16 Bit)
                        ;                     (Partial Straight Line Version)
                        ;
                        ;     This routine converts a 16 Bit binary Number to a 5
                        ; BCD Number.
                        ;
                        ;       The 16 bit binary number is input in locations Hbyt
                        ; Lbyte with the high byte in Hbyte.
                        ;       The 5 digit BCD number is returned in R0, R1 and R2
                        ; containing the MSD in its right most nibble.
                        ;
                        ;   Performance :
                        ;                Program Memory  :  44
                        ;                Clock Cycles    :  572
                        ;
                        ;*******************************************************
                        ;
                        B2_BCD_Straight
003F 8404                 bsf     _fs0
0040 8504                 bsf     _fs1            ; set fsr0 for no auto inc
                        ;
0041 8804                 bcf     _carry
0042 2925                 clrf    count
0043 8425                 bsf     count,4         ; set count = 16
0044 2924                 clrf    R0
0045 2923                 clrf    R1
0046 2922                 clrf    R2
                        loop16b
0047 1B20                 rlcf    Lbyte
0048 1B21                 rlcf    Hbyte
0049 1B22                 rlcf    R2
004A 1B23                 rlcf    R1
004B 1B24                 rlcf    R0
                        ;
004C 2725                 dcfsnz  count
004D 0002                 return                  ; DONE
004E B022                 movlw   R2              ; load R2 as indirect addr
004F 0101                 movwf   fsr0
                        ; adjustBCD
0050 6000                 movfp   indf0,wreg
0051 B103                 addlw   0x03
0052 9B00                 btfsc   wreg,3          ; test if result > 7
0053 0100                 movwf   indf0
0054 6000                 movfp   indf0,wreg
0055 B130                 addlw   0x30
0056 9F00                 btfsc   wreg,7          ; test if result > 7
0057 0100                 movwf   indf0           ; save as MSD
                        ;
0058 1501                 incf    fsr0
                        ; adjustBCD
0059 6000                 movfp   indf0,wreg
```


© 1994 Microchip Technology Incorporated

```
005A B103                addlw    0x03
005B 9B00                btfsc    wreg,3          ; test if result > 7
005C 0100                movwf    indf0
005D 6000                movfp    indf0,wreg
005E B130                addlw    0x30
005F 9F00                btfsc    wreg,7          ; test if result > 7
0060 0100                movwf    indf0           ; save as MSD
                       ;
0061 1501                incf     fsr0
                       ; adjustBCD
0062 6000                movfp    indf0,wreg
0063 B103                addlw    0x03
0064 9B00                btfsc    wreg,3          ; test if result > 7
0065 0100                movwf    indf0
0066 6000                movfp    indf0,wreg
0067 B130                addlw    0x30
0068 9F00                btfsc    wreg,7          ; test if result > 7
0069 0100                movwf    indf0           ; save as MSD
                       ;
006A C047                goto     loop16b
                       ;
                       ;*******************************************************
                       ;                BCD To Binary Conversion
                       ;
                       ;      This routine converts a 5 digit BCD number to a 16
                       ; number.
                       ;      The input 5 digit BCD numbers are asumed to be in 1
                       ; R0, R1 & R2 with R0 containing the MSD in its right most
                       ;
                       ;      The 16 bit binary number is output in registers Hby
                       ; ( high byte & low byte repectively ).
                       ;
                       ;      The method used for conversion is :
                       ;                input number X = abcde ( the 5 digit BCD nu
                       ;        X = (R0,R1,R2) = abcde = 10[10[10[10a+b]+c]+d]+e
                       ;
                       ;   Performance :
                       ;                Program Memory  :  30
                       ;                Clock Cycles    :  112
                       ;
                       ;*******************************************************

                       ;
                       mpy10b
006B B50F                andlw    0x0f
006C 0F20                addwf    Lbyte
006D 9804                btfsc    _carry
006E 1521                incf     Hbyte
                       mpy10a
006F 8804                bcf      _carry          ; multiply by 2
0070 1A20                rlcf     Lbyte,w
0071 0127                movwf    Ltemp
0072 1A21                rlcf     Hbyte,w         ; (Htemp,Ltemp) = 2*N
0073 0126                movwf    Htemp
                       ;
0074 8804                bcf      _carry          ; multiply by 2
0075 1B20                rlcf     Lbyte
0076 1B21                rlcf     Hbyte
0077 8804                bcf      _carry          ; multiply by 2
0078 1B20                rlcf     Lbyte
0079 1B21                rlcf     Hbyte
007A 8804                bcf      _carry          ; multiply by 2
007B 1B20                rlcf     Lbyte
007C 1B21                rlcf     Hbyte           ; (Hbyte,Lbyte) = 8*N
                       ;
007D 6027                movfp    Ltemp,wreg
007E 0F20                addwf    Lbyte
007F 6026                movfp    Htemp,wreg
0080 1121                addwfc   Hbyte
```


© 1994 Microchip Technology Incorporated

```
0081 0002                    return                  ; (Hbyte,Lbyte) = 10*N
                             ;
                             ;
                             BCDtoB
0082 2921                    clrf      Hbyte
0083 6024                    movfp     R0,wreg
0084 B50F                    andlw     0x0f
0085 0120                    movwf     Lbyte
0086 E06F                    call      mpy10a          ; result = 10a+b
                             ;
0087 1C23                    swapf     R1,w
0088 E06B                    call      mpy10b          ; result = 10[10a+b]
                             ;
0089 6023                    movfp     R1,wreg
008A E06B                    call      mpy10b          ; result = 10[10[10a+b]+c]

                             ;
008B 1C22                    swapf     R2,w
008C E06B                    call      mpy10b          ; result = 10[10[10[10a+b]+
                             ;
008D 6022                    movfp     R2,wreg
008E B50F                    andlw     0x0f
008F 0F20                    addwf     Lbyte
0090 9804                    btfsc     _carry
0091 1521                    incf      Hbyte           ; result = 10[10[10[10a+b]

0092 0002                    return                    ; BCD to binary conversion
                             ;
                             ;*******************************************************
                             ;
                             ;                 Unsigned BCD Addition
                             ;
                             ;       This routine performs a 2 Digit Unsigned BCD Additi
                             ; It is assumed that the two BCD numbers to be added are in
                             ; locations Num1 & Num2. The result is the sum of Num1+Num2
                             ; and is stored in location Num2 and the overflow carry is
                             ; in location Num1
                             ;
                             ;   Performance :
                             ;                Program Memory  :       5
                             ;                Clock Cycles    :       5
                             ;
                             ;*******************************************************
                             ;
                             BCDAdd
0093 6026                    movfp     Num1,wreg
0094 0E27                    addwf     Num2,w          ; perform binary addition

0095 2F27                    daw       Num2              ; adjust for BCD addition
0096 2926                    clrf      Num1
0097 1B26                    rlcf      Num1              ; set Num1 = carry bit
0098 0002                    return
                             ;
                             ;*******************************************************
                             ;
                                 END


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Incorporated

APPENDIX C: FXP_DIV.ASM

```
;       PIC17 FIXED POINT DIVIDE ROUTINES       VERSION 1.5
;       Input:  fixed point arguments in AARG and BARG
;       Output: quotient AARG/BARG followed by remainder in REM
;       All timings are worst case cycle counts
;       It is useful to note that the additional routines FXD3115U, FXD1515U, and
;       FXD1507U can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.
;         Routine            Clocks     Function
;       FXD3216S    414 32 bit/16 bit -> 32.16 signed fixed point divide
;       FXD3216U    485 32 bit/16 bit -> 32.16 unsigned fixed point divide
;       FXD3215U    390 32 bit/15 bit -> 32.15 unsigned fixed point divide
;       FXD3115U    383 31 bit/15 bit -> 31.15 unsigned fixed point divide
;       FXD1616S    214 16 bit/16 bit -> 16.16 signed fixed point divide
;       FXD1616U    244 16 bit/16 bit -> 16.16 unsigned fixed point divide
;       FXD1615U    197 16 bit/15 bit -> 16.15 unsigned fixed point divide
;       FXD1515U    191 15 bit/15 bit -> 15.15 unsigned fixed point divide
;       FXD1608S    146 16 bit/08 bit -> 16.08 signed fixed point divide
;       FXD1608U    196 16 bit/08 bit -> 16.08 unsigned fixed point divide
;       FXD1607U    130 16 bit/07 bit -> 15.07 unsigned fixed point divide
;       FXD1507U    125 15 bit/07 bit -> 15.07 unsigned fixed point divide
                list    r=dec,x=on,t=off,p=17C42
                include <PIC17.INC>
;*****************************************************************************************
;*****************************************************************************************
;       Define divide register variables
ACC             equ     0x19    ; most significant byte of contiguous 4 byte accumulator
SIGN            equ     0x1F    ; save location for sign in MSB
TEMP            equ     0x25    ; temporary storage
;       Define binary operation arguments
AARG            equ     0x19    ; most significant byte of argument A
BARG            equ     0x22    ; most significant byte of argument B
REM             equ     0x1D    ; most significant byte of remainder
;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )
;*****************************************************************************************
;*****************************************************************************************
;       32 Bit Division Macros
SDIV3216        macro
;       Max Timing:     5+8+30*12+6 = 379 clks
;       Min Timing:     5+8+30*11+6 = 349 clks
;       PM: 5+8+30*14+6 = 439           DM: 8
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                  i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
```

© 1994 Microchip Technology Incorporated

```
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD268
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK268
SADD268         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0

SOK268          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD2616
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK2616
SADD2616        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK2616 RLCF                    ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
```


© 1994 Microchip Technology Incorporated

```
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            SADD2624
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK2624
SADD2624        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK2624         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            SADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK26#v(i)
SADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            SOK26
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK26
                endm
UDIV3216  macro
;       restore = 15/20 clks,  nonrestore = 11/14 clks
;       Max Timing: 16*15+1+16*20 = 561 clks
;       Min Timing: 16*11+1+16*14 = 401 clks
;       PM:  16*15+1+16*20 = 561                DM: 9
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
```


© 1994 Microchip Technology Incorporated

```
                i = 8
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                CLRF            TEMP
                i = 16
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                i = 24
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK26#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                endm
```


© 1994 Microchip Technology Incorporated

```
NDIV3216        macro
;       Max Timing:     10+31*15+6 = 481 clks
;       Min Timing: 10+31*14+6 = 450 clks
;       PM: 10+31*19+6 = 605            DM: 9
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
        BTFSS                   ACC+B0,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD268
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK268
NADD268         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK268          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
```


© 1994 Microchip Technology Incorporated

```
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD2616
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK2616
NADD2616        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK2616         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            NADD2624
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK2624
NADD2624        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
```


© 1994 Microchip Technology Incorporated

```
                CLRF            WREG
                ADDWFC          TEMP
NOK2624         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            NADD26#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK26#v(i)
NADD26#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK26#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            NOK26
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
NOK26
                endm
UDIV3215        macro
;       Max Timing:     8+31*12+6 = 386 clks
;       Min Timing:     8+31*11+6 = 355 clks
;       PM: 8+31*14+6 = 448             DM: 8
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
```


© 1994 Microchip Technology Incorporated

```
                BTFSS           ACC+B0,LSB
                GOTO            UADD258
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK258
UADD258         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK258          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD2516
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK2516
UADD2516        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK2516         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD2524
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
```


© 1994 Microchip Technology Incorporated

```
                GOTO            UOK2524
UADD2524        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK2524         RLCF            ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            UADD25#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK25#v(i)
UADD25#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            UOK25
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK25
                endm
UDIV3115        macro
;       Max Timing:     5+8+30*12+6 = 379 clks
;       Min Timing:     5+8+30*11+6 = 349 clks
;       PM: 5+8+30*14+6 = 439           DM: 8
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
```


© 1994 Microchip Technology Incorporated

```
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD158
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK158
UADD158         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK158          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD1516
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK1516
UADD1516        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK1516         RLCF            ACC+B2
                i = 17
                while i < 24
                RLCF            ACC+B2,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B2
                i=i+1
                endw
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B2,LSB
                GOTO            UADD1524
                SUBWF           REM+B1
```


© 1994 Microchip Technology Incorporated

```
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK1524
UADD1524        ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK1524 RLCF                    ACC+B3
                i = 25
                while i < 32
                RLCF            ACC+B3,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B3,LSB
                GOTO            UADD15#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK15#v(i)
UADD15#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15#v(i)      RLCF            ACC+B3
                i=i+1
                endw
                BTFSC           ACC+B3,LSB
                GOTO            UOK15
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK15
                endm
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Division Macros
SDIV1616        macro
;       Max Timing:     5+8+14*12+6 = 187 clks
;       Min Timing:     5+8+14*11+6 = 173 clks
;       PM: 5+8+14*14+6 = 215           DM: 6
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
        BTFSS                   ACC+B0,LSB
                GOTO            SADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK66#v(i)
SADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
```

© 1994 Microchip Technology Incorporated

```
SOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            SADD668
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK668
SADD668         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK668          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            SADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            SOK66#v(i)
SADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            SOK66
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
SOK66
                endm
UDIV1616  macro
;       restore = 15 clks,  nonrestore = 11 clks
;       Max Timing: 8*15+8*15 = 240 clks
;       Min Timing: 8*11+8*11 = 176 clks
;       PM: 8*15+8*15 = 240             DM: 6
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                i = 8
```


© 1994 Microchip Technology Incorporated

```
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             _C
UOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                endm
NDIV1616        macro
;       Max Timing:     9+15*15+6 = 240 clks
;       Min Timing:     9+15*14+6 = 225 clks
;       PM: 9+15*19+6 = 300             DM: 7
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK66#v(i)

NADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK66#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD668
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
```


© 1994 Microchip Technology Incorporated

```
                GOTO            NOK668
NADD668         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK668          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD66#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK66#v(i)
NADD66#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK66#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            NOK66
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
NOK66
                endm
UDIV1615        macro
;       Max Timing:     7+15*12+6 = 193 clks
;       Min Timing:     7+15*11+6 = 178 clks
;       PM: 7+15*14+6 = 213             DM: 6
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B1
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                i = 1
               while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD65#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK65#v(i)
UADD65#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65#v(i)      RLCF            ACC+B0
                i=i+1
               endw
                RLCF            ACC+B1,W
```


© 1994 Microchip Technology Incorporated

```
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD658
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK658
UADD658         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK658          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B1,LSB
                GOTO            UADD65#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK65#v(i)
UADD65#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            UOK65
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK65
                endm
UDIV1515        macro
;       Max Timing:     5+8+14*12+6 = 187 clks
;       Min Timing:     5+8+14*11+6 = 173 clks
;       PM: 5+8+14*14+6 = 215           DM: 6
                variable i
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD55#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK55#v(i)
```


© 1994 Microchip Technology Incorporated

```
UADD55#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                BTFSS           ACC+B0,LSB
                GOTO            UADD558

                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK558
UADD558         ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK558          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG

                BTFSS           ACC+B1,LSB
                GOTO            UADD55#v(i)
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                GOTO            UOK55#v(i)
UADD55#v(i)     ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSC           ACC+B1,LSB
                GOTO            UOK55
                MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
UOK55
                endm
;_____________________________________________________________________________
;       Extra 16 Bit Divide Macros
DIV1616         macro
;       Timing: restore = 16 clks,  nonrestore = 13 clks        16*16 = 256 clks
                variable i
                i = 0
                while i < 16
                RLCF            AARG+B1
                RLCF            AARG+B0
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BTFSS           _C
                GOTO            RS1616_#v( i )
                BSF             AARG+B1,LSB
                GOTO            OK1616_#v( i )
RS1616_#v( i )  MOVFP           BARG+B1,WREG
                ADDWF           REM+B1
```


© 1994 Microchip Technology Incorporated

```
                MOVFP           BARG+B0,WREG
                ADDWFC          REM+B0
                BCF             AARG+B1,LSB
OK1616_#v(i)
                i=i+1
                endw
                endm
DIVMAC          macro
;       Timing: restore = 19 clks,  nonrestore = 14 clks      16*19 = 304 clks
                variable i
                i = 0
                while i < 16
                RLCF            AARG+B1
                RLCF            AARG+B0
                RLCF            REM+B1
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0,W
                BTFSS           _Z
                GOTO            notz#v( i )
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1,W
notz#v( i )     BTFSS           _C
                GOTO            nosub#v( i )
                MOVFP           BARG+B1,WREG
                SUBWF           REM+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          REM+B0
                BSF             AARG+B1,LSB
                GOTO            ok#v(i)
nosub#v(i)      BCF             AARG+B1,LSB
ok#v(i)
                i=i+1
                endw
                endm
;*****************************************************************************************
;*****************************************************************************************
;       16/08 BIT Division Macros
SDIV1608        macro
;       Max Timing:     3+5+14*8+2 = 122 clks
;       Min Timing:     3+5+14*8+2 = 122 clks
;       PM: 3+5+14*8+2 = 122            DM: 4
                variable i
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
```


© 1994 Microchip Technology Incorporated

```
                ADDWF           REM+B0
                RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
UDIV1608  macro
;       restore = 9/15 clks,  nonrestore = 8/11 clks
;       Max Timing: 8*9+1+8*15 = 193 clks       max
;       Min Timing: 8*8+1+8*11 = 153 clks       min
;       PM: 8*9+1+8*15 = 193            DM: 4
                variable         i
                i = 0
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                BTFSC           _C
                GOTO            UOK68#v(i)
                ADDWF           REM+B0
                BCF             _C
UOK68#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                CLRF            TEMP
                i = 8
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                BTFSC           _C
                GOTO            UOK68#v(i)
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
                BCF             _C
UOK68#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                endm
NDIV1608        macro
;       Max Timing:     7+15*12+3 = 190 clks
;       Min Timing: 7+15*11+3 = 175 clks
;       PM: 7+15*14+3 = 220             DM: 5
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                CLRF            TEMP,W
                SUBWFB          TEMP
                RLCF            ACC+B0
```


© 1994 Microchip Technology Incorporated

```
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD68#v(i)
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK68#v(i)
NADD68#v(i)     ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK68#v(i)      RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B0,LSB
                GOTO            NADD688
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK688
NADD688         ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP
NOK688          RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                RLCF            TEMP
                MOVFP           BARG+B0,WREG
                BTFSS           ACC+B1,LSB
                GOTO            NADD68#v(i)
                SUBWF           REM+B0
                CLRF            WREG
                SUBWFB          TEMP
                GOTO            NOK68#v(i)
NADD68#v(i)     ADDWF           REM+B0
                CLRF            WREG
                ADDWFC          TEMP

NOK68#v(i)      RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                endm
UDIV1607        macro
;       Max Timing:     5+15*8+2 = 127 clks
;       Min Timing:     5+15*8+2 = 127 clks
;       PM: 5+15*8+2 = 127              DM: 4
                variable i
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                i = 1
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
```

```
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0

                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                 i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
UDIV1507        macro
;       Max Timing:     3+5+14*8+2 = 122 clks
;       Min Timing:     3+5+14*8+2 = 122 clks
;       PM: 3+5+14*8+2 = 122            DM: 4
                variable i
                MOVFP           BARG+B0,WREG
                SUBWF           REM+B0
                RLCF            ACC+B0
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                ADDWF           REM+B0
                RLCF            ACC+B0
                i = 2
                while i < 8
                RLCF            ACC+B0,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B0
                i=i+1
                endw
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i = 9
                while i < 16
                RLCF            ACC+B1,W
                RLCF            REM+B0
                MOVFP           BARG+B0,WREG
```


© 1994 Microchip Technology Incorporated

```
                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLCF            ACC+B1
                i=i+1
                endw
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                endm
;*****************************************************************************************
;*****************************************************************************************
;       32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16
;       Input:  32 bit signed fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3216S
;       Output: 32 bit signed fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-   AARG / BARG
;       Max Timing:     11+379+3 = 393 clks             A > 0, B > 0
;                               14+379+17 = 410 clks            A > 0, B < 0
;                               18+379+17 = 414 clks            A < 0, B > 0
;                               21+379+3 = 403 clks             A < 0, B < 0
;       Min Timing:     11+349+3 = 363 clks             A > 0, B > 0
;                               14+349+17 = 380 clks            A > 0, B < 0
;                               18+349+17 = 384 clks            A < 0, B > 0
;                               21+349+3 = 373 clks             A < 0, B < 0
;       PM: 21+439+16 = 476             DM: 9
FXD3216S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0
                CLRF            REM+B1,W
                BTFSS           BARG+B0,MSB     ; if MSB set go & negate BARG
                GOTO            CA3216S
                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B1
                ADDWFC          BARG+B0
CA3216S         BTFSS           AARG+B0,MSB     ; if MSB set go & negate ACCa
                GOTO            C3216S
                COMF            AARG+B3
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B3
                ADDWFC          AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
C3216S  SDIV3216
                BTFSS           SIGN,MSB        ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B3
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B3
                ADDWFC          AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B1
                COMF            REM+B0
                INCF            REM+B1
                ADDWFC          REM+B0
                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************
```

4

© 1994 Microchip Technology Incorporated

```
;       32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16
;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3216U
;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1AARG+B2,AARG+B3
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+481+2 = 485 clks
;       Min Timing:     2+450+2 = 459 clks
;       PM: 2+605+1 = 608               DM: 9
FXD3216U        CLRF            REM+B0
                CLRF            REM+B1
                NDIV3216
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       32/15 Bit Unsigned Fixed Point Divide 32/15 -> 32.15

;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3215U
;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+386+2 = 390 clks
;       Min Timing:     2+355+2 = 359 clks
;       PM: 2+448+1 = 451               DM: 8
FXD3215U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV3215
                 RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15
;       Input:  31 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD3115U
;       Output: 31 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+379+2 = 383 clks
;       Min Timing:     2+349+2 = 353 clks
;       PM: 2+439+1 = 442               DM: 8
FXD3115U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV3115
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16
;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               16 bit fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1616S
;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               16 bit fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     11+187+3 = 201 clks             A > 0, B > 0
;                                  14+187+13 = 214 clks         A > 0, B < 0
;                                  14+187+13 = 214 clks         A < 0, B > 0
;                                  17+187+3 = 207 clks          A < 0, B < 0
;       Min Timing:     11+173+3 = 187 clks             A > 0, B > 0
;                                  14+173+13 = 200 clks         A > 0, B < 0
;                                  14+173+13 = 200 clks         A < 0, B > 0
;                                  17+173+3 = 193 clks          A < 0, B < 0
;       PM: 14+215+12 = 241             DM: 7
FXD1616S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0
                CLRF            REM+B1,W
```


© 1994 Microchip Technology Incorporated

```
                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA1616S

                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B1
                ADDWFC          BARG+B0
CA1616S         BTFSS           AARG+B0,MSB     ; if MSB set go & negate ACCa
                GOTO            C1616S
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B1
                ADDWFC          AARG+B0
C1616S  SDIV1616
                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B1
                COMF            REM+B0
                INCF            REM+B1
                ADDWFC          REM+B0
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1616U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+240+2 = 244 clks
;       Min Timing:     2+176+2 = 180 clks
;       PM: 2+240+1 = 243               DM: 6
FXD1616U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1616
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/15 Bit Unsigned Fixed Point Divide 16/15 -> 16.15
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;       Use:    CALL    FXD1615U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+193+2 = 197 clks
;       Min Timing:     2+178+2 = 182 clks
;      PM: 2+213+1 = 216               DM: 6
FXD1615U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1615
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15
;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1
;      Use:     CALL    FXD1515U
;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     2+187+2 = 191 clks
;       Min Timing:     2+173+2 = 177 clks
```

```
;       PM: 2+215+1 = 218                       DM: 6
FXD1515U        CLRF            REM+B0
                CLRF            REM+B1
                UDIV1515
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/8 Bit Signed Fixed Point Divide 16/08 -> 16.08
;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               8 bit fixed point divisor in BARG+B0
;       Use:    CALL    FXD1608S
;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               8 bit fixed point remainder in REM+B0
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     10+122+3 = 135 clks             A > 0, B > 0
;                                  11+122+11 = 144 clks           A > 0, B < 0
;                                  13+122+11 = 146 clks           A < 0, B > 0
;                                  14+122+3 = 139 clks            A < 0, B < 0
;       Min Timing:     10+122+3 = 135 clks             A > 0, B > 0
;                                  11+122+11 = 144 clks           A > 0, B < 0
;                                  13+122+11 = 146 clks           A < 0, B > 0
;                                  14+122+3 = 139 clks            A < 0, B < 0
;       PM: 14+122+10 = 146                     DM: 5
FXD1608S        MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVWF           SIGN
                CLRF            REM+B0,W
                BTFSS           BARG+B0,MSB             ; if MSB set go & negate BARG
                GOTO            CA1608S
                COMF            BARG+B0
                INCF            BARG+B0
CA1608S         BTFSS           AARG+B0,MSB             ; if MSB set go & negate ACCa
                GOTO            C1608S
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B1
                ADDWFC          AARG+B0
C1608S  SDIV1608
                BTFSS           SIGN,MSB                ; negate (ACCc,ACCd)
                RETLW           0x00
                COMF            AARG+B1
                COMF            AARG+B0
                CLRF            WREG
                INCF            AARG+B1
                ADDWFC          AARG+B0
                COMF            REM+B0
                INCF            REM+B0
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/8 Bit Unsigned Fixed Point Divide 16/08 -> 16.08
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                8 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1608U
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               8 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     1+193+2 = 196 clks
;       Min Timing:     1+153+2 = 156 clks
;       PM: 1+193+1 = 195               DM: 4
FXD1608U        CLRF            REM+B0
                UDIV1608
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       16/7 Bit Unsigned Fixed Point Divide 16/07 -> 16.07
;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                7 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1607U
```


© 1994 Microchip Technology Incorporated

```
;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     1+127+2 = 130 clks
;       Min Timing:     1+127+2 = 130 clks
;       PM: 1+127+1 = 129               DM: 4
FXD1607U        CLRF            REM+B0
                UDIV1607
                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************
;       15/7 Bit Unsigned Fixed Point Divide 15/07 -> 15.07
;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;                7 bit unsigned fixed point divisor in BARG+B0
;       Use:    CALL    FXD1507U
;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0
;       Result: AARG, REM  <-  AARG / BARG
;       Max Timing:     1+122+2 = 125 clks
;       Min Timing:     1+122+2 = 125 clks
;      PM: 1+122+1 = 124             DM: 4
FXD1507U        CLRF            REM+B0
                UDIV1507
                RETLW           0x00
                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1994 Microchip Technology Incorporated

© 1994 Microchip Technology Incorporated

MICROCHIP

AN540

Implementing IIR Digital Filters

INTRODUCTION

This application note describes the implementation of various digital filters using the PIC17C42, the first member of Microchip's 2nd generation 8-bit microcontrollers. The PIC17C42 is a very high speed 8-bit microcontroller with an instruction cycle time of 250ns (@ 16 MHz input clock). Even though PIC17C42 is an 8-bit device, it's high speed and efficient instruction set allows implementation of digital filters for practical applications. Traditionally digital filters are implemented using expensive Digital Signal Processors (DSPs). In a system the DSP is normally a slave processor being controlled by either an 8- or 16-bit microcontroller. Where sampling rates are not high (esp. in mechanical control systems), a single chip solution is possible using the PIC17C42.

This application note provides a few examples of implementing digital filters. Example code for 2nd order Infinite Impulse Response (IIR) filters is given. The following type of filters are implemented:

- Low Pass
- High Pass
- Band Pass
- Band Stop (notch) filter

This application note does not explain how to design a filter. Filter design theory is well established and is beyond the scope of this application note. It is assumed that a filter is designed according to the desired specifications. The desired digital filters may be designed using either standard techniques or using commonly available digital filter design software packages.

Finite Impulse Response (FIR) filters have many advantages over IIR filters, but are much more resource intensive (both in terms of execution time and RAM). On the other hand, IIR filters are quite attractive for implementing with the PIC17C42 resources. Especially where phase information is not so important, IIR filters are a good choice (FIR filters have a linear phase response). Of the various forms used for realizing digital filters (like, Direct form, Direct II form, Cascade form, Parallel, Lattice structure, etc.) the Direct II form is used in this application note. It is easy to understand and simple macros can be built using these structures.

THEORY OF OPERATION

Digital filters in most cases assume the following form of relationship between the output and input sequences.

$$y(n) = -\sum_{i=o}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

The above equation basically states that the present output is a weighted sum of the past inputs and past outputs. In case of FIR filters, the weighted constants ai=0 and in case of IIR filters, at least one of the ai constant is non zero. In case of IIR, the above formula may be re written in terms of Z transform as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k Z^{-k}}{1 + \sum_{k=1}^{N} a_k Z^{-k}}$$

The above equation can further be rewritten in difference equation format as follows:

$$y(n) = -\sum_{i=1}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

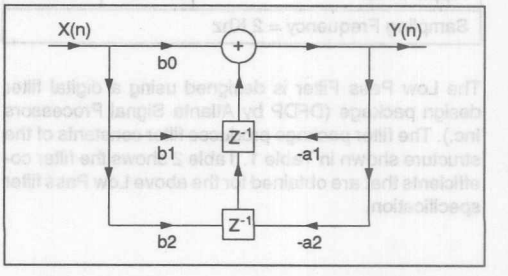
Realization of the above equation is called as the Direct Form II structure. For example, in case of a second order structure, M=N=2, gives the following difference equations :

$$d(n) = x(n) + a_1 d(n-1) + a_2 d(n-2) \qquad (1)$$

$$y(n) = b_0 d(n) + b_1 d(n-1) + b_2 (d(n-2) \qquad (2)$$

The above difference equations may be represented as shown in Figure 1.

FIGURE 1 - 2ND ORDER DIRECT FORM II STRUCTURE (TRANSPOSED)



4

© 1993 Microchip Technology Inc.

The structure as shown in Figure 1 may be cascaded to attain a higher order filter. For example, if two stages are cascaded together, a 4th Order IIR Filter is obtained. This way, the output of the 1st stage becomes the input to the second stage. Multiple order filters are thus implemented by cascading a 2nd order filter structure as shown in Figure 1.

IMPLEMENTATION

A 4th order IIR Filter is implemented by cascading two structures shown in Figure 1. The output Y (output of each filter stage) is computed by direct implementation of Equations (1) & (2). Since each stage is similar algorithmically, it is implemented as a Macro using Microchip's Assembler/Linker for PIC17C42. This Macro (labelled **"BIQUAD"**) is called twice for implementing a 4th order filter. The output of the 1st stage is directly fed to the input of the second stage without any scaling.

Scaling is required depending on the particular application. The user can modify the code very easily without any penalty on speed. Also, saturation arithmetic is not used. Overflows can be avoided by limiting the input sequence amplitude. All numbers are assumed to be 16 bits in Q15 format (15 decimal points, MSB is sign bit). Thus the user must scale and sign extend the input sequence accordingly. For example, if the input is from a 12-bit A/D converter, the user must sign extend the 12-bit input if bit 11 is a one.

The BIQUAD macro is a generic macro and can be used for all IIR filters whether it is Low Pass, High Pass, Band Pass or Band Stop. A general purpose 16x16 multiplier routine is also provided. This routine is implemented as a straight line code for speed considerations.

The 4th order IIR filter implemented is a Low Pass Filter with the specifications as shown in Table 1.

TABLE 1 - FILTER CONSTANTS

	BAND1	BAND2
Lower Band Edge	0.0	600 Hz
Upper Band Edge	500 Hz	1 Khz
Nominal Gain	1.0	0.0
Nominal Ripple	0.01	0.05
Maximum Ripple	0.00906	0.04601
Ripple in dB	0.07830	-26.75
Sampling Frequency = 2 Khz		

The Low Pass Filter is designed using a digital filter design package (DFDP by Atlanta Signal Processors Inc.). The filter package produces filter constants of the structure shown in Table 1. Table 2 shows the filter co-efficients that are obtained for the above Low Pass filter specification.

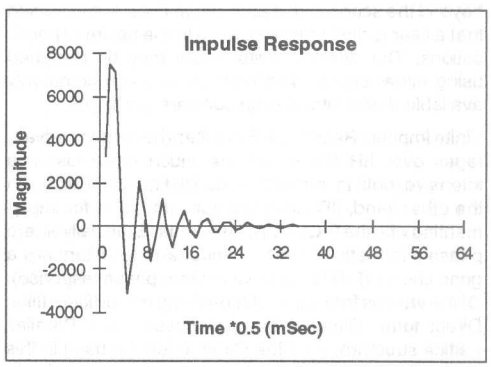
TABLE 2 - FILTER COEFFICIENTS

Co-efficients	a1	a2	b0	b1	b2
Stage 1	-0.133331	0.167145	0.285431	0.462921	0.285431
Stage 2	0.147827	0.765900	0.698273	0.499908	0.698273

The above filter co-efficients (5 per stage) are quantized to Q15 format (i.e they are multiplied by 32768) and saved in program memory (starting at label **"_coeff_lpass"**). The constants for both the stages are read into data memory using TLRD and TABLRD instructions in the Initialization Routine (labelled **"initFilter"**). The user may read the coefficients of only one stage at a time and save some RAM at the expense of speed.

The sample 4th order Low Pass IIR Filter is tested by analyzing the impulse response of the filter. An impulse signal is fed as input to the filter. This is simulated by forcing the input to the filter by a large quantity (say 7F00h) on the first input sample, and the all zeros from the 2nd sample onwards. The output sequence is the filter's impulse response and is captured into the PICMASTER's (Microchip's Universal In-Circuit Emulator) real time trace buffer. This captured data from PICMASTER is saved to file and analyzed. Analysis was done using MathCad for Windows and DSP Analysis program from Burr-Brown (DSPLAY). The Fourier Transform of this Impulse response of the filter should display the Filter's frequency response, in this case being a Low Pass type. The plots of the impulse response and the frequency response are shown in figures below.

FIGURE 2 - IMPULSE RESPONSE CAPTURED FROM PIC-MASTER



DSPLAY is a trademark of Burr-Brown
DFDP is a trademark of Atlanta Signal Processing Inc.
Windows is a trademark of Microsoft Corporation
MathCad is a registered trademark of MathSoft, Inc.

© 1993 Microchip Technology Inc.

FIGURE 3 - SPECTRUM COMPUTED FROM IMPULSE RESPONSE

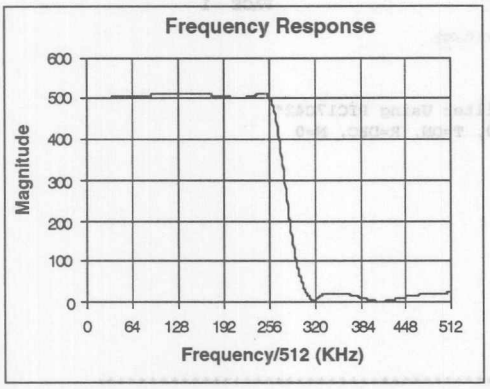
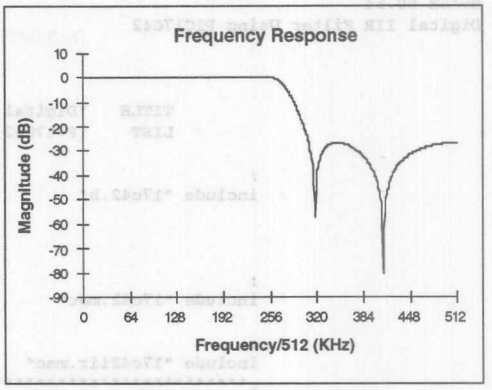


FIGURE 4 - LOG MAGNITUDE SPECTRUM OF IMPULSE RESPONSE



4

PERFORMANCE

The resource requirements for filter implementations using PIC17C42 is given below. These numbers can be used to determine whether a higher order filter can be executed in real time. The same information may be used to determine the highest sampling rate possible.

FILTER APPLICATIONS

Digital filters find applications in many areas especially involving processing of real world signals. In some applications like ABS systems in an automobile, digital filtering becomes a must. In this case elimination of noise (especially glitches and false readings of sensors) is very critical and thus the requirement of digital signal processing.

Digital filters are also needed in Process Control where notch filters and low pass filters are desired because the signals from sensors are transmitted over long lines, especially in a very noisy environment. In this case typically a notch filter (centering 50Hz or 60Hz) is used. In case of eliminating background noise, a band stop filter (e.g. 40Hz to 120Hz) is used. The sample code given in this application note can be used to design a feedback control system's digital compensator. For example, a typical DC Motor's digital compensator (like dead beat compensator) is of second order and has the same filter structure as is implemented in this application note.

TABLE 3 - RESOURCE REQUIREMENTS

Timing (Cycles)†	#of Filter Stages*775 + 16
Program Memory (locations)†	#of Filter Stages*68 + 290
RAM (File Registers)	#of Filter Stages*16+16

†: The above numbers do not include the initialization routine.

Author: Amar Palacherla
Logic Products Division

TABLE 4 - RESOURCE REQUIREMENTS

Filter Order	Cycles	Real Time (@ 16 MHz)	Maximum Sampling	Program Memory†	RAM
2	791	197.75 uSec	5.05 Khz	358	32
4	1566	391.5 uSec	2.55 Khz	426	48
6	2341	585.25 uSec	1.7 Khz	494	64
8	3116	779.0 uSec	1.28 Khz	562	80
10	3891	972.75 uSec	1.0 Khz	630	96

†: The above numbers do not include the initialization routine.

© 1993 Microchip Technology Inc.

APPENDIX A: IIR.LST

```
MPASM B0.54                                                    PAGE  1
Digital IIR Filter Using PIC17C42


                        TITLE   "Digital IIR Filter Using PIC17C42"
                        LIST     P=17C42, C=120, T=ON, R=DEC, N=0

                    ;
                    include "17c42.h"



                    ;
                    include "17c42.mac"

                    ;
                    include "17c42iir.mac"
                    ;*****************************************************************
                    ;                       PIC17C42 MACRO
                    ;
                    ;               Macro For A Bi-Quad IIR Filter
                    ;         2nd order Direct Form (Transposed) Type
                    ;
                    ; Filter co-efficients B0 & B2 are assumed equal
                    ;
                    ; The difference equations for each cascade section is given by :
                    ;             Y(n) = B0*D(n) + B1*D(n-1) + B2*D(n-2)
                    ;             D(n) = X(n) - A1*D(n-1) -  A2*D(n-2)
                    ;      where X(n) = input sample, Y(n) = output of filter
                    ;      and A1, A2, B0, B1, B2 are the Filter Co-efficients
                    ;
                    ;  The above difference equations are only for 1 section of a
                    ; 2nd order Direct_Form II Filter structure (IIR)
                    ;
                    ;   NOTE :
                    ;             It is possible to design the above structures
                    ;        such that the co-efficients B0 = B2. If this is the
                    ;        case,
                    ;                 Y(n) = B0*[D(n) + D(n-2)] = B2*[D(n) + D(n-2)]
                    ;        This way, one multiplication can be avoided
                    ;
                    ;  If a 4th order filter is to be implemented, the output of
                    ; the 1st structure should be input to the 2nd cascade section
                    ;
                    ;       Timing (WORST CASE) :
                    ;                               59+4*179 = 775 Cycles
                    ;                               (194 uS @ 16 Mhz)
                    ;       Program Memory  :
                    ;                               63 locations
                    ;
                    ;*****************************************************************
                    ;       The sample filters are desined so that B0=B2
                    ;       This saves 1 multiplication
                    ;
0001                B0_EQUALS_B2    equ     TRUE
                    ;
                    ;*****************************************************************
                    ;  Parameters to BIQUAD Macro :
                    ;               Filter Constants A1, A2, B0, B1, B2
                    ;               & D(n), D(n-1), D(n-2), filter stage #
                    ;
                    BIQUAD  MACRO   Ax1,Ax2,Bx0,Bx1,Dn,Dn_1,Dn_2,stage

                    ;
                    ; Compute Ax2*D(n-2)
```


© 1993 Microchip Technology Inc.

```
;
MOVFP16  Dn_2,AARG              ; D(n-2) = multiplier
MOVFP16  Ax2,BARG               ; A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = A2*D(n-2)
;
; Add product to output of 1st section
; Save result in 32 bit Accumulator
;
ADD32    DPX,ACC
;
;  Compute A1*D(n-1)
;
MOVFP16  Dn_1,AARG              ; AARG = D(n-2) = multiplier
MOVFP16  Ax1,BARG               ; BARG = A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = A1*D(n-1)
;
; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
;       multiplications already done, so simply perform a 32 bit add
;       of previously obtained multiplication results
;
ADD32   DPX,ACC                 ; ACC = A1*D(n-1)+A2*D(n-2)+(output of 1st
;
;
; save the upper 16 bits  of D(n) from the 32 bit accumulator
; left shift the result by 1, to adjust the decimal point after
; a Q15*Q15 multiplication
;
rlcf     ACC+B1,w
rlcf     ACC+B2,w
movwf    Dn
rlcf     ACC+B3,w               ; decimal adjust ( mult by 2)
movwf    Dn+B1
;
; Compute B2 * [D(n) + D(n-2)]
;
if B0_EQUALS_B2
ADD16ACC Dn_2,Dn,AARG           ; AARG = Dn + D(n-2) = multiplier
MOVFP16  Bx0,BARG               ; BARG = A2 = multiplicand
call     DblMult                ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
MOVPF32  DPX,ACC

else
MOVFP16  Bx0,BARG
MOVFP16  Dn,AARG
call     DblMult                ; B0*D(n)
MOVPF32  DPX,ACC

MOVFP16  Bx2,BARG
MOVFP16  Dn_2,AARG
call     DblMult                ; B2*D(n-2)
ADD32    DPX,ACC
endif
;
; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
; This way in the next iteration D(n-2) is equal to the present D(n-1)
;
movfp    Dn_1,AARG+B0
movpf    AARG+B0,Dn_2           ; Shift down D(n-1)
movfp    Dn_1+B1,AARG+B1
movpf    AARG+B1,Dn_2+B1        ; AARG = D(n-1) = multiplier
MOVFP16  Bx1,BARG               ; BARG = B1 = multiplicand
call     DblMult                ; (ACCd,ACCc) = B1*D(n-1)
;
; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
;                  = B1*D(n-1) + B0*[D(n) + D(n-2)]
; Since all multiplications are already done, simply perform a
; 32 bit addition
;
ADD32   DPX,ACC                 ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
```

4

© 1993 Microchip Technology Inc.

```
;
; Shift down D(n) to D(n-1) so that in the next iteration, the new
; D(n-1) is the present D(n)
;
MOV16    Dn,Dn_1             ; Shift down D(n) to D(n-1)
;
         ENDM


;
;*****************************************************************
;               Second Order Direct Form IIR Filter
;
; In the code given below, a 4th order IIR Elliptic Lowpass Filter
; is implemented. Other order filters may be implemented by
; taking the following example code as a basis.
;
;       The specifications of the filter are :
;
;       Sampling Frequency = 2.0 Khz
;
;       Filter Type = 4th Order Elliptic Lowpass Filter
;                                   Band1           Band2
;       Lower Band Edge             0.0             600 Hz
;       Upper Band Edge             500 Hz          1 Khz
;       Nominal Gain                1.0             0.0
;       Nominal Ripple              0.01            0.05
;       Maximum Ripple              0.00906         0.04601
;       Ripple in dB                0.07830         -26.75
;
;  The Filter Co-efficients for the above specifications
; of the filter are computed as follows :
;
;      1st Section :
;                          A11 = -0.133331
;                          A12 = 0.167145
;                          B10 = 0.285431
;                          B11 = 0.462921
;                          B12 = 0.285431
;      2nd Section
;                          A21 = 0.147827
;                          A22 = 0.765900
;                          B20 = 0.698273
;                          B21 = 0.499908
;                          B22 = 0.698273
;
;
;       Performance (WORST Case):
;
;                          Cycles    = #of Filter Stages*775 + 16
;                                    = 2*775+16 = 1566 Cycles
;                                          ( 391 uSec)
;                                    per each sample. Initialization
;                                    time after reset is not counted
;                                    Timing measured with B0_EQUALS_B2
;                                    set to TRUE (see BIQUAD Macro for
;                                    explanation
;
;                          Program Memory :
;                                    = 16+ # of FilterStages * (BIQUAD
;                                            + filter co-efficients)
;                                            + multiplier
;                                    = 16+2*(63+5)+274 = 421 locations
;                                      (excluding initialization)
;
;                          RAM usage = 48 file registers
;                          RAM usage/each additional stage = 16 file regs
;
```


© 1993 Microchip Technology Inc.

```
                        ;
                        ;                       This time is less than 2 Khz (500 uSec),
                        ;                       which means real time filtering is possible
                        ;
                        ;*****************************************************************

                        ;*****************************************************************
                        ;
                        ;
                                CBLOCK  0
0000 0004                       B0,B1,B2,B3
                                ENDC
                        ;*****************************************************************
                        ;
                                CBLOCK  0x18
0018 0004                       DPX,DPX1,DPX2,DPX3          ; arithmetic accumulator
001C 0004                       AARG,AARG1,BARG,BARG1       ; multiply arguments
                                ENDC
                        ;
                                CBLOCK
0020 0002                       Dn1, Dn1_Hi
0022 0002                       Dn1_1, Dn1_1_Hi
0024 0002                       Dn1_2, Dn1_2_Hi
0026 0000
0026 0002                       Dn2, Dn2_Hi
0028 0002                       Dn2_1, Dn2_1_Hi
002A 0002                       Dn2_2, Dn2_2_Hi
                                ENDC
                                CBLOCK
002C 0002                       A11, A11_Hi
002E 0002                       A12, A12_Hi
0030 0002                       B10, B10_Hi
0032 0002                       B11, B11_Hi             ; 1st Section Filter Co-efficients
0034 0002                       B12, B12_Hi
0036 0000
0036 0002                       A21, A21_Hi
0038 0002                       A22, A22_Hi
003A 0002                       B20, B20_Hi
003C 0002                       B21, B21_Hi             ; 2nd Section Filter Co-efficients
003E 0002                       B22, B22_Hi
                                ENDC

                                CBLOCK
0040 0002                       X, X1                   ; 16 bits of input stream
0042 0002                       Y, Y1                   ; 16 bits of filter output
0044 0000
0044 0004                       ACC, ACC1, ACC2, ACC3   ; 32 bit accumulator for computations
                                ENDC
                        ;
0002                    FltStage        .set    2
000A                    NumCoeff        equ     (5*FltStage) ; 5 Co-eff per stage
                        ;
0001                    LPASS   .set    TRUE
0000                    HPASS   .set    FALSE
0000                    BPASS   .set    FALSE               ; select the desired filter type
0000                    BSTOP   .set    FALSE
                        ;
0001                    SIGNED  equ     TRUE                ; Set This To 'TRUE' for signed multi
                        ;                                   ; and 'FALSE' for unsigned.
                        ;
                        ;*****************************************************************
                        ;               Test Program For Low Pass Filter
                        ;*****************************************************************

                                ORG     0x0000
                        ;
                        start
0000 E00D                       call    initFilter
```

```
0001 B000                       movlw   0x00
0002 0140                       movwf   X
0003 B07F                       movlw   0x7f                    ; set initial Xn = X(0) = 0x7f00
0004 0141                       movwf   X+B1                    ; test for impulse response
                        ;
                        NextPoint
0005 E022                       call    IIR_Filter
0006 A442                       tlwt    _LOW,Y
                        tracePoint
0007 AE43                       tablwt  _HIGH,0,Y+B1
0008 0000                       nop
0009 2940                       clrf    X                       ; set X(n) = 0 , n <> 0
000A 2941                       clrf    X+B1                    ; for simulating an Impulse
000B C005                       goto    NextPoint
                        ;
000C C00C               self    goto    self
                        ;
                        ;*****************************************************************
                        ;

                        initFilter

                        ;
                        ;   At first read the Filter Co-efficients from Prog. Mem to Data RAM
                        ;
                                if      LPASS
000D B0C1                               movlw   _coeff_lpass
000E 010D                               movwf   tblptrl
000F B001                               movlw   page    _coeff_lpass
0010 010E                               movwf   tblptrh
                                endif
                                if      HPASS
                                        movlw   _coeff_hpass
                                        movwf   tblptrl
                                        movlw   page    _coeff_hpass
                                        movwf   tblptrh
                                endif
                                if      BPASS
                                        movlw   _coeff_bpass
                                        movwf   tblptrl
                                        movlw   page    _coeff_bpass
                                        movwf   tblptrh
                                endif
                                if      BSTOP
                                        movlw   _coeff_bstop
                                        movwf   tblptrl
                                        movlw   page    _coeff_bstop
                                        movwf   tblptrh
                                endif

                        ;
0011 B02C                       movlw   A11
0012 0101                       movwf    fsr0
0013 8404                       bsf     _fs0
0014 8D04                       bcf     _fs1                    ; auto increment
                        ;
                        ; Read Filter Co-efficients from Program Memory
                        ;
0015 B00A                       movlw   NumCoeff
0016 A92C                       tablrd  _LOW,_INC,A11           ; garbage
                        NextCoeff
0017 A000                       tlrd    _LOW,indf0
0018 AB00                       tablrd  _HIGH,_INC,indf0
0019 1700                       decfsz  wreg
001A C017                       goto    NextCoeff
                        ;
                        ; Initilize "Dn"s to zero
                        ;
001B B020                       movlw   Dn1
```


© 1993 Microchip Technology Inc.

```
001C 0101                       movwf   fsr0
001D B00C                       movlw   6*FltStage
                        NextClr
001E 2900                       clrf    indf0
001F 1700                       decfsz  wreg
0020 C01E                       goto    NextClr
                        ;
0021 0002                       return
                        ;*****************************************************************
                        ;                       1st Cascade Section
                        ;*****************************************************************
                        ;
                        IIR_Filter

                        ; Compute  D(n) = X(n) + A1*D(n-1) + A2*D(n-2)
                        ;          Since the filter constants are computated in Q15 format,
                        ;          X(n) must be multiplied by 2**15 and then added to the
                        ;          other terms.
                        ;
                        ; Move Input to accumulator after proper scaling
                        ;
0022 8804                       bcf     _carry
0023 1941                       rrcf    X+B1
0024 1940                       rrcf    X
0025 2900                       clrf    wreg                    ; Scale the input X
0026 1900                       rrcf    wreg
0027 0145                       movwf   ACC+B1
0028 6040                       movfp   X,wreg
0029 0146                       movwf   ACC+B2
002A 6041                       movfp   X+B1,wreg
002B 0147                       movwf   ACC+B3                  ; ACC = scaled input : X*(2**15)
                        ;
                        ; 1st Biquad filter section
                        ;
                        BIQUAD  A11,A12,B10,B11,Dn1,Dn1_1,Dn1_2,1

                        ;
                        ; Compute A12*D(n-2)
                        ;

002C 7C24                       MOVFP   Dn1_2+B0,AARG+B0        ; move Dn1_2(B0) to AARG(B0)
002D 7D25                       MOVFP   Dn1_2+B1,AARG+B1        ; move Dn1_2(B1) to AARG(B1)


002E 7E2E                       MOVFP   A12+B0,BARG+B0          ; move A12(B0) to BARG(B0)
002F 7F2F                       MOVFP   A12+B1,BARG+B1          ; move A12(B1) to BARG(B1)

0030 E0AF                       call    DblMult         ; (ACCd,ACCc) = A2*D(n-2)
                        ;
                        ; Add product to output of 1st section
                        ; Save result in 32 bit Accumulator
                        ;
0031 6018                       MOVFP   DPX+B0,WREG     ; get lowest byte of DPX into w
0032 0F44                       ADDWF   ACC+B0          ; add lowest byte of ACC, save in ACC(B0
0033 6019                       MOVFP   DPX+B1,WREG     ; get 2nd byte of DPX into w
0034 1145                       ADDWFC  ACC+B1          ; add 2nd byte of ACC, save in ACC(B1)
0035 601A                       MOVFP   DPX+B2,WREG     ; get 3rd byte of DPX into w
0036 1146                       ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
```

© 1993 Microchip Technology Inc.

```
0037 601B                   MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0038 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ;  Compute A1*D(n-1)
                    ;


0039 7C22                   MOVFP   Dn1_1+B0,AARG+B0        ; move Dn1_1(B0) to AARG(B0)
003A 7D23                   MOVFP   Dn1_1+B1,AARG+B1        ; move Dn1_1(B1) to AARG(B1)


003B 7E2C                   MOVFP   A11+B0,BARG+B0          ; move A11(B0) to BARG(B0)
003C 7F2D                   MOVFP   A11+B1,BARG+B1          ; move A11(B1) to BARG(B1)

003D E0AF                   call    DblMult                 ; (ACCd,ACCc) = A1*D(n-1)
                    ;
                    ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                    ;       multiplications already done, so simply perform a 32 bit add
                    ;       of previously obtained multiplication results
                    ;


003E 6018                   MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
003F 0F44                   ADDWF   ACC+B0                  ; add lowest byte of ACC,save in ACC(B0)
0040 6019                   MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0041 1145                   ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0042 601A                   MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0043 1146                   ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0044 601B                   MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0045 1147                   ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                    ;
                    ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                    ; left shift the result by 1, to adjust the decimal point after
                    ; a Q15*Q15 multiplication
                    ;
0046 1A45                   rlcf    ACC+B1,w
0047 1A46                   rlcf    ACC+B2,w
0048 0120                   movwf   Dn1
0049 1A47                   rlcf    ACC+B3,w                ; decimal adjust ( mult by 2)
004A 0121                   movwf   Dn1+B1
                    ;
                    ; Compute B2 * [D(n) + D(n-2)]
                    ;
                            if B0_EQUALS_B2


004B 6024                   movfp   Dn1_2+B0,wreg
004C 0E20                   addwf   Dn1+B0,w
004D 011C                   movwf   AARG+B0
004E 6025                   movfp   Dn1_2+B1,wreg
004F 1021                   addwfc  Dn1+B1,w
0050 011D                   movwf   AARG+B1
0051 7E30                   MOVFP   B10+B0,BARG+B0          ; move B10(B0) to BARG(B0)
0052 7F31                   MOVFP   B10+B1,BARG+B1          ; move B10(B1) to BARG(B1)
0053 E0AF                   call    DblMult                 ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
0054 5844                   MOVPF   DPX+B0,ACC+B0           ; move DPX(B0) to ACC(B0)
0055 5945                   MOVPF   DPX+B1,ACC+B1           ; move DPX(B1) to ACC(B1)
0056 5A46                   MOVPF   DPX+B2,ACC+B2           ; move DPX(B2) to ACC(B2)
0057 5B47                   MOVPF   DPX+B3,ACC+B3           ; move DPX(B3) to ACC(B3)
```


© 1993 Microchip Technology Inc.

```
                        else
                                MOVFP16 B10,BARG
                                MOVFP16 Dn1,AARG
                                call    DblMult         ; B0*D(n)
                                MOVPF32 DPX,ACC
                                MOVFP16 Bx2,BARG
                                MOVFP16 Dn1_2,AARG
                                call    DblMult         ; B2*D(n-2)
                                ADD32   DPX,ACC
                         endif
                    ;
                    ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                    ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                    ;
0058 7C22                movfp   Dn1_1,AARG+B0
0059 5C24                movpf   AARG+B0,Dn1_2           ; Shift down D(n-1)
005A 7D23                movfp   Dn1_1+B1,AARG+B1
005B 5D25                movpf   AARG+B1,Dn1_2+B1        ; AARG = D(n-1) = multiplier
005C 7E32                MOVFP   B11+B0,BARG+B0          ; move B11(B0) to BARG(B0)
005D 7F33                MOVFP   B11+B1,BARG+B1          ; move B11(B1) to BARG(B1)
005E E0AF                call    DblMult                 ; (ACCd,ACCc) = B1*D(n-1)
                    ;
                    ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                    ;                   = B1*D(n-1) + B0*[D(n) + D(n-2)]
                    ; Since all multiplications are already done, simply perform a
                    ; 32 bit addition
                    ;
005F 6018                MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
0060 0F44                ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
0061 6019                MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0062 1145                ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0063 601A                MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0064 1146                ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0065 601B                MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0066 1147                ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)
                    ;
                    ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                    ; D(n-1) is the present D(n)
                    ;
0067 6020                MOVFP   Dn1+B0,WREG             ; get byte of Dn1 into w
0068 0122                MOVWF   Dn1_1+B0                ; move to Dn1_1(B0)
0069 6021                MOVFP   Dn1+B1,WREG             ; get byte of Dn1 into w
006A 0123                MOVWF   Dn1_1+B1                ; move to Dn1_1(B1)
                    ;
                    ;
                    ; 2nd Biquad filter section
                    ;
                    BIQUAD  A21,A22,B20,B21,Dn2,Dn2_1,Dn2_2,2
                    ;
                    ; Compute A22*D(n-2)
                    ;
006B 7C2A                MOVFP   Dn2_2+B0,AARG+B0        ; move Dn2_2(B0) to AARG(B0)
006C 7D2B                MOVFP   Dn2_2+B1,AARG+B1        ; move Dn2_2(B1) to AARG(B1) 006D 7E38
                         MOVFP   A22+B0,BARG+B0          ; move A22(B0) to BARG(B0)
006E 7F39                MOVFP   A22+B1,BARG+B1          ; move A22(B1) to BARG(B1)
006F E0AF                call    DblMult                 ; (ACCd,ACCc) = A2*D(n-2)
                    ;
                    ; Add product to output of 1st section
                    ; Save result in 32 bit Accumulator
                    ;

0070 6018                MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
0071 0F44                ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
0072 6019                MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0073 1145                ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0074 601A                MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0075 1146                ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0076 601B                MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
```

4

© 1993 Microchip Technology Inc.

```
0077 1147                ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)

                         ;
                         ;  Compute A1*D(n-1)
                         ;


0078 7C28                MOVFP   Dn2_1+B0,AARG+B0        ; move Dn2_1(B0) to AARG(B0)
0079 7D29                MOVFP   Dn2_1+B1,AARG+B1        ; move Dn2_1(B1) to AARG(B1)


007A 7E36                MOVFP   A21+B0,BARG+B0          ; move A21(B0) to BARG(B0)
007B 7F37                MOVFP   A21+B1,BARG+B1          ; move A21(B1) to BARG(B1)

007C E0AF                call    DblMult                 ; (ACCd,ACCc) = A1*D(n-1)
                         ;
                         ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                         ;  multiplications already done, so simply perform a 32 bit add
                         ;  of previously obtained multiplication results
                         ;


007D 6018                MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
007E 0F44                ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B0)
007F 6019                MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
0080 1145                ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
0081 601A                MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
0082 1146                ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
0083 601B                MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
0084 1147                ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)
                         ;
                         ;
                         ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                         ; left shift the result by 1, to adjust the decimal point after
                         ; a Q15*Q15 multiplication
                         ;
0085 1A45                rlcf    ACC+B1,w
0086 1A46                rlcf    ACC+B2,w
0087 0126                movwf   Dn2
0088 1A47                rlcf    ACC+B3,w                ; decimal adjust ( mult by 2)
0089 0127                movwf   Dn2+B1
                         ;
                         ; Compute B2 * [D(n) + D(n-2)]
                         ;
                         if      B0_EQUALS_B2


008A 602A                        movfp   Dn2_2+B0,wreg
008B 0E26                        addwf   Dn2+B0,w
008C 011C                        movwf   AARG+B0
008D 602B                        movfp   Dn2_2+B1,wreg
008E 1027                        addwfc  Dn2+B1,w
008F 011D                        movwf   AARG+B1


0090 7E3A                        MOVFP   B20+B0,BARG+B0          ; move B20(B0) to BARG(B0)
0091 7F3B                        MOVFP   B20+B1,BARG+B1          ; move B20(B1) to BARG(B1)

0092 E0AF                        call    DblMult         ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
0093 5844                        MOVPF   DPX+B0,ACC+B0   ; move DPX(B0) to ACC(B0)
0094 5945                        MOVPF   DPX+B1,ACC+B1   ; move DPX(B1) to ACC(B1)
0095 5A46                        MOVPF   DPX+B2,ACC+B2   ; move DPX(B2) to ACC(B2)
0096 5B47                        MOVPF   DPX+B3,ACC+B3   ; move DPX(B3) to ACC(B3)

                         else
```


© 1993 Microchip Technology Inc.

```
                        MOVFP16 B20,BARG
                        MOVFP16 Dn2,AARG
                        call    DblMult         ; B0*D(n)
                        MOVPF32 DPX,ACC
                        MOVFP16 Bx2,BARG
                        MOVFP16 Dn2_2,AARG
                        call    DblMult         ; B2*D(n-2)
                        ADD32   DPX,ACC
                endif
                ;
                ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                ;
0097 7C28               movfp   Dn2_1,AARG+B0
0098 5C2A               movpf   AARG+B0,Dn2_2           ; Shift down D(n-1)
0099 7D29               movfp   Dn2_1+B1,AARG+B1
009A 5D2B               movpf   AARG+B1,Dn2_2+B1        ; AARG = D(n-1) = multiplier
009B 7E3C               MOVFP   B21+B0,BARG+B0          ; move B21(B0) to BARG(B0)
009C 7F3D               MOVFP   B21+B1,BARG+B1          ; move B21(B1) to BARG(B1)
009D E0AF               call    DblMult                 ; (ACCd,ACCc) = B1*D(n-1)
                ;
                ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                ;                   = B1*D(n-1) + B0*[D(n) + D(n-2)]
                ; Since all multiplications are already done, simply perform a
                ; 32 bit addition
                ;
009E 6018               MOVFP   DPX+B0,WREG             ; get lowest byte of DPX into w
009F 0F44               ADDWF   ACC+B0                  ; add lowest byte of ACC, save in ACC(B
00A0 6019               MOVFP   DPX+B1,WREG             ; get 2nd byte of DPX into w
00A1 1145               ADDWFC  ACC+B1                  ; add 2nd byte of ACC, save in ACC(B1)
00A2 601A               MOVFP   DPX+B2,WREG             ; get 3rd byte of DPX into w
00A3 1146               ADDWFC  ACC+B2                  ; add 3rd byte of ACC, save in ACC(B2)
00A4 601B               MOVFP   DPX+B3,WREG             ; get 4th byte of DPX into w
00A5 1147               ADDWFC  ACC+B3                  ; add 4th byte of ACC, save in ACC(B3)
                ;
                ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                ; D(n-1) is the present D(n)
                ;
00A6 6026               MOVFP   Dn2+B0,WREG             ; get byte of Dn2 into w
00A7 0128               MOVWF   Dn2_1+B0                ; move to Dn2_1(B0)
00A8 6027               MOVFP   Dn2+B1,WREG             ; get byte of Dn2 into w
00A9 0129               MOVWF   Dn2_1+B1                ; move to Dn2_1(B1)
                ;
                ;
                ; The filter output is now computed
                ; Save the Upper 16 Bits of 32 bit Accumulator into Y after
                ; adjusting the decimal point
                ;
                MOV16   ACC+B2,Y
00AA 6046               MOVFP   ACC+B2+B0,WREG          ; get byte of ACC+B2 into w
00AB 0142               MOVWF   Y+B0                    ; move to Y(B0)
00AC 6047               MOVFP   ACC+B2+B1,WREG          ; get byte of ACC+B2 into w
00AD 0143               MOVWF   Y+B1                    ; move to Y(B1)
                ;
00AE 0002               return                          ; Output Y(n) computed
                ;*****************************************************************
                ;  Set SIGNED/UNSIGNED Flag Before Including 17c42MPY.mac
                ;
                include         "17c42MPY.mac"
                ;*****************************************************************
                ;  Low Pass Filter Co-efficients
                ;
                ;
```

4

```
                    ;                       ELLIPTIC     LOWPASS FILTER
                    ;
                    ;                 FILTER ORDER =    4
                    ;                 Sampling frequency =   2.000  KiloHertz
                    ;
                    ;                        BAND  1        BAND  2
                    ;
                    ;
                    ; LOWER BAND EDGE          .00000         .60000
                    ; UPPER BAND EDGE          .50000        1.00000
                    ; NOMINAL GAIN            1.00000         .00000
                    ; NOMINAL RIPPLE           .01000         .05000
                    ; MAXIMUM RIPPLE           .00906         .04601
                    ; RIPPLE IN dB             .07830      -26.74235
                    ;
                    ;  I        A(I,1)         A(I,2)         B(I,0)         B(I,1)         B(I,2)
                    ;
                    ;  1      -.133331         .167145        .285431        .462921        .285431
                    ;  2       .147827         .765900        .698273        .499908        .698273
                    ;

                    _coeff_lpass                          ; co-efficients for 1st Cascade
01C1 1111                 data     4369                   ; -A11
01C2 EA9B                 data     -5477                  ; -A12
01C3 2489                 data     9353                   ; B10
01C4 3B41                 data     15169                  ; B11
01C5 2489                 data     9353                   ; B12
                    ;                                     ; co-efficients for 2nd Cascade
01C6 ED14                 data    -4844                   ; -A21
01C7 9DF7                 data    -25097                  ; -A22
01C8 5961                 data     22881                  ; B20
01C9 3FFD                 data     16381                  ; B21
01CA 5961                 data     22881                  ; B22
                    ;
                    ;*****************************************************************
                    ;
                    ;
                    ;*****************************************************************
                    ;  High Pass Filter Co-efficients
                    ;
                    ;                       ELLIPTIC    HIGHPASS FILTER
                    ;
                    ;                 FILTER ORDER =    4
                    ;                 Sampling frequency =   2.000  KiloHertz
                    ;
                    ;                        BAND  1        BAND  2
                    ;
                    ;
                    ; LOWER BAND EDGE          .00000         .50000
                    ; UPPER BAND EDGE          .40000        1.00000
                    ; NOMINAL GAIN             .00000        1.00000
                    ; NOMINAL RIPPLE           .04000         .02000
                    ; MAXIMUM RIPPLE           .03368         .01686
                    ; RIPPLE IN dB          -29.45335         .14526
                    ;
                    ;  I        A(I,1)         A(I,2)         B(I,0)         B(I,1)         B(I,2)
                    ;
                    ;  1       .276886        .195648        .253677       -.411407        .253677
                    ;  2      -.094299        .780396        .678650       -.485840        .678650
                    ;
                    ;
                    _coeff_hpass                          ; co-efficients for 1st Cascade section
01CB DC8F                 data     -9073                  ; -A11
01CC E6F5                 data     -6411                  ; -A12
01CD 2079                 data     8313                   ; B10
01CE CB57                 data     -13481                 ; B11
01CF 2079                 data     8313                   ; B12
                    ;                                     ; co-efficients for 2nd Cascade section
```


© 1993 Microchip Technology Inc.

```
01D0 0C12                data    3090            ; -A21
01D1 9C1C                data    -25572          ; -A22
01D2 56DE                data    22238           ; B20
01D3 C1D0                data    -15920          ; B21
01D4 56DE                data    22238           ; B22
                         ;
                         ;*****************************************************************
                         ;
                         ;*****************************************************************
                         ;  Band Pass Filter Co-efficients
                         ;
                         ;
                         ;                    ELLIPTIC      BANDPASS FILTER
                         ;
                         ;                 FILTER ORDER =    4
                         ;                 Sampling frequency =    2.000  KiloHertz
                         ;
                         ;                       BAND  1        BAND  2        BAND  3
                         ;
                         ;
                         ; LOWER BAND EDGE         .00000         .30000         .90000
                         ; UPPER BAND EDGE         .10000         .70000        1.00000
                         ; NOMINAL GAIN            .00000        1.00000         .00000
                         ; NOMINAL RIPPLE          .05000         .05000         .05000
                         ; MAXIMUM RIPPLE          .03644         .03867         .03641
                         ; RIPPLE IN dB          -28.76779         .32956      -28.77647
                         ;
                         ;
                         ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                         ;
                         ;  1       -.936676       .550568       .444000       -.865173       .444000
                         ;  2        .936707       .550568       .615540       1.199402       .615540
                         ;

                         _coeff_bpass                        ; co-efficients for 1st Cascade section
01D5 3BF2                data     30693/2           ; -A11
01D6 DCC4                data     -18041/2          ; -A12
01D7 1C6A                data     14549/2           ; B10
01D8 C8A1                data     -28350/2          ; B11
01D9 1C6A                data     14549/2           ; B12
                         ;                          ; co-efficients for 2nd Cascade section
01DA C40D                data    -30694/2           ; -A21
01DB DCC4                data    -18041/2           ; -A22
01DC 2765                data     20170/2           ; B20
01DD 4CC3                data     39302/2           ; B21
01DE 2765                data     20170/2           ; B22
                         ;
                         ;*****************************************************************
                         ;
                         ;*****************************************************************
                         ;  Band Stop Filter Co-efficients
                         ;
                         ;
                         ;                      ELLIPTIC     BANDSTOP FILTER
                         ;
                         ;                FILTER ORDER =     4
                         ;                Sampling frequency =    2.000  KiloHertz
                         ;
                         ;                       BAND  1        BAND  2        BAND  3
                         ;
                         ;
                         ;LOWER BAND EDGE         .00000         .45000         .70000
                         ;UPPER BAND EDGE         .30000         .55000        1.00000
                         ;NOMINAL GAIN           1.00000         .00000        1.00000
                         ;NOMINAL RIPPLE          .05000         .05000         .05000
                         ;MAXIMUM RIPPLE          .03516         .03241         .03517
                         ;RIPPLE IN dB            .30015      -29.78523         .30027
                         ;
                         ;
```

4

© 1993 Microchip Technology Inc.

```
                    ;  I        A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                    ;
                    ;  1        .749420       .583282       .392685       .087936       .392685
                    ;  2       -.749390       .583282      1.210022      -.270935      1.210022
                    ;
                    _coeff_bstop                        ; co-efficients for 1st Cascade section
01DF D00A                   data     -24557/2           ; -A11
01E0 DAAC                   data     -19113/2           ; -A12
01E1 1922                   data      12868/2           ; B10
01E2 05A0                   data      2881/2            ; B11
01E3 1922                   data      12868/2           ; B12
                                                        ; co-efficients for 2nd Cascade section
01E4 2FF6                   data     24556/2            ; -A21
01E5 DAAC                   data     -19113/2           ; -A22
01E6 4D71                   data     39650/2            ; B20
01E7 EEA9                   data     -8878/2            ; B21
01E8 4D71                   data     39650/2            ; B22
                    ;
                    ;*****************************************************************

                            END

 Errors   :     0
 Warnings :     0
```


© 1993 Microchip Technology Inc.

MICROCHIP

AN542

Implementation of Fast Fourier Transforms

INTRODUCTION

Fourier transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent and analyze discrete-time signals. However, direct implementation of DFT is computationally very inefficient. Of the various available high speed algorithms to compute DFT, the Cooley-Tukey algorithm is the simplest and most commonly used. These efficient algorithms to compute DFTs are called Fast Fourier Transforms (FFTs).

This application note provides the source code to compute the FFT using PIC17C42. The theory behind the FFT algorithms is well established and described in the literature and hence not described in this application note. A Radix-2 Cooley- Tukey FFT is implemented with no limits on the length of FFT. The length is only limited by the amount of available program memory space. All computations are done using double precision arithmetic.

IMPLEMENTATION

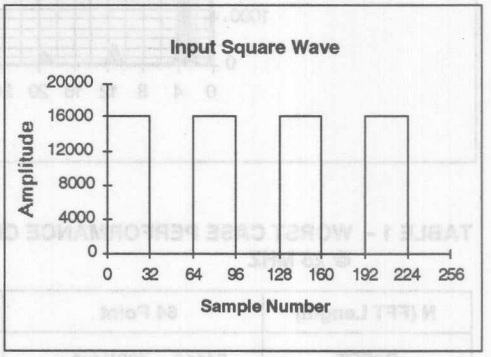
Since the PIC17C42 has only 232 x 8 general purpose RAM (equivalent of 116 x 16), at most a 32 point FFT (16-bit REAL & IMAGINARY data) can be implemented using on chip RAM. To compute higher point FFT, the data is stored in program memory space of PIC17C42. The PIC17C42 has instructions (TABLRD & TABLWT) to transfer data between program memory space and on chip file registers. In extended microcontroller mode, the PIC17C42 has 2K x 16 (0000h:07FFh) on chip program memory space and 62K x 16 (0800h:0FFFFh) of external program memory space. In this mode, the code (in this case, the FFT code) may reside on the on chip EPROM and the data to be analyzed may be stored in external RAMs (up to 62K). A suggested method of connecting external RAMs (appropriate EEPROMs may also be used) is shown in Figure 3.

If PIC17C42 is used in extended microcontroller mode and if all the code resides on chip, then the cost may further be reduced by using only one external SRAM instead of two. The block diagram is shown in Figure 2. The 16-bit data stored in the external RAM is organized as low byte followed by high byte. To achieve this, the code presented in this application note needs minor modifications, especially where TABLRD and TABLWR instructions are used. Address indexing must be incremented by two since 2 reads/writes must be performed to access a 16-bit data.

The FFT is implemented with Decimation In Frequency. Thus the input data before calling the FFT routine ("R2FFT") should be in normal order and the transformed data is in scrambled order. The original data is overwritten by the transformed data to conserve memory. This is achieved by use of in-place calculations. This in-place calculations causes the order of the DFT terms to be permuted. So at the end of the transform, all the data needs to be unscrambled to get the right order of the DFT terms. In some applications the order of terms is not necessary. Keeping this in mind, the unscrambling code is written as a separate subroutine ("Unscramble") and may be called if necessary.

Before implementing the FFT using PIC17C42, a "C" program was written and tested. This high level programming helps in writing the assembly code and the results of both programs can be compared against while debugging the assembly code. The "C" source code for the Radix-2 FFT is shown in Appendix A. The assembly code source file of the FFT program is shown in Appendix B. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note AN00540.

FIGURE 1 - TEST WAVE FORM



4

© 1993 Microchip Technology Inc.

TESTING

The assembly code was developed and debugged using Microchip's PICMASTER™ In-Circuit Emulator System. A main program generates a test pattern (like a square wave) and calls the FFT routines "R2FFT" & "Unscramble". After the DFT terms are computed, the results are captured into PICMASTER's real time trace buffer by putting a trace point on a dummy TABLRD instruction and capturing only the 2nd cycle (data cycle of TABLRD) of the instruction. The data from Trace buffer was hot linked to Microsoft Excel™ using DDE and then the graphs were plotted and analyzed.

The code was tested on various wave forms (a rectangular pulse, a triangular wave, square wave and a sine wave) and using FFT lengths of 64, 256 & 1024. The results of a 256 Point FFT on a square wave is shown below. The test wave form is shown in Figure 1 and the frequency spectrum of it computed by PIC17C42 is shown in Figure 2. As expected, the spectra appears at the odd harmonics of the input waveform's fundamental frequency (At N*256/64, N = 0,1,3,5,..).

PERFORMANCE

The performance of FFT using PIC17C42 is quite impressive noting that for an 8-bit machine with no hardware multiplier. Also note that all computations are performed using double precision arithmetic (16- and 32- bits) which is the case in most of the low end DSPs. Table 1 provides the real time performance in total number of Instruction cycles for both the "R2FFT" and "Unscramble" routines using 64, 256 and 1024 Point FFT. Note that the timings are in a worst case situation and in general will be a lot better than shown in the table. The worst case situation arises because the 16 x 16 software multiplier ("DblMult") does not have a uniform timing and depends on the input data. The worst case timing of the multiplier is used in the computation of performance.

FIGURE 2 - FFT (MAGNITUDE SPECTRUM) OF FIGURE 1 COMPUTED BY PIC17C42

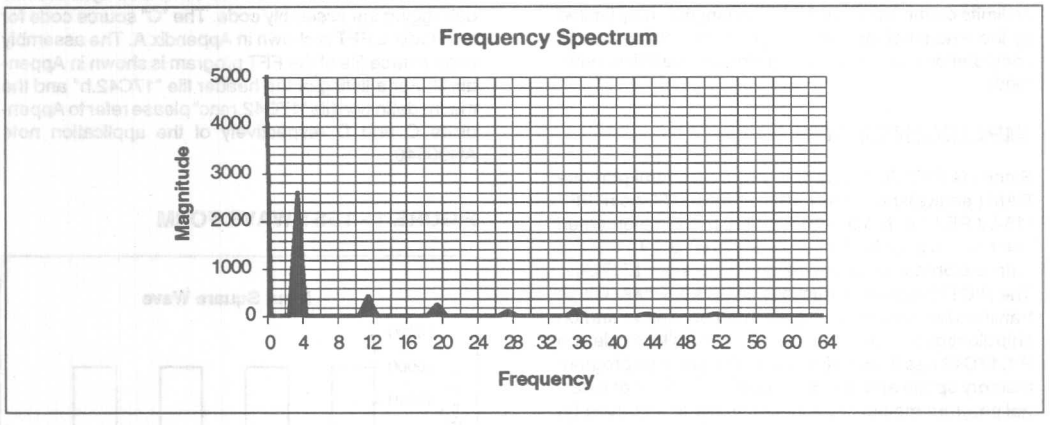


TABLE 1 - WORST CASE PERFORMANCE OF FFT IN INSTRUCTION CYCLES AND REAL TIME @ 25 MHZ

N (FFT Length)	64 Point	256 Point	1024 Point
R2FFT	34116 + 768*Mult = 171588	178024 + 4096*Mult= 911208	878752 + 20480*Mult = 4544672
Unscramble	5143	22495	93525
Total	176731 (28.28 mS)	933703 (149.39 mS)	4638197 (742.11 mS)

© 1993 Microchip Technology Inc.

Table 2 shows the Program Memory and Data RAM requirements for an N Point FFT. The multiplier routine and other general purpose macro requirements are included in the memory requirements. The speed performance for the square wave test data differs from Table 1 since "worst case timings" is not used, and reflects a more reasonable data.

FFT APPLICATIONS

Although the FFT does not find a place in many microcontroller applications, it is very useful in providing a benchmark of the processor. As can be seen from table 2, the performance is very satisfactory, considering the fact that PIC17C42 is a Microcontroller and not a DSP. Also it should be borne in mind that all computations are performed in 16/32 bit arithmetic and that PIC17C42 is a low-cost 8-bit device unlike the DSPs which are relatively expensive.

In applications like Instrumentation, where real time FFT computation is not required, PIC17C42 can be used as a single chip solution instead of a Microcontroller and a Digital Signal Processor.

Suggested Reading :

[1] Rabiner. L.R., and Gold, B., Theory and Application Of Digital Signal Processing, Englewood Cliffs, NJ : Prentice-Hall, 1975.

[2] Burrs, C.S., and Parks, T.W., DFT/FFT and Convolution Alogorithms, New York : Wiley, 1985.

[3] Rodriguez,Jeffrey J., "An Improved FFT Digit-Reversal Algorithm," IEEE Transactions On Acoustics, Speech, And Signal Processing, Vol. 37, No. 8, Aug 1989.

Author: Amar Palacherla
Logic Products Division

4

TABLE 2 - REQUIREMENTS FOR RADIX-2 FFT @ 25 MHZ

N (FFT Length)	64 Point	256 Point	1024 Point
Code Space (locations)	603 + 0.75*N = 651	603 + 0.75*N = 795	603 + 0.75*N= 1371
Data Storage in Program Memory Space	2*N = 128	2*N = 512	2*N = 2048
Scratch RAM	49	49	49
Performance (Square Wave Input Data)	122384 (19.58 mS)	644416 (103.11 mS)	3192176 (510.75mS)

FIGURE 3 - EXTERNAL MEMORY CONNECTION

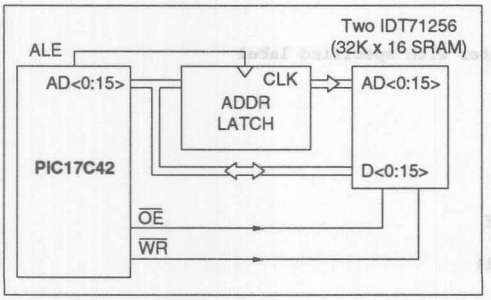
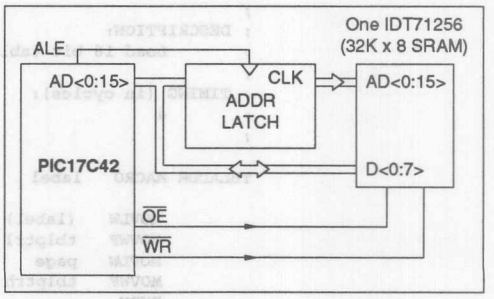


FIGURE 4 - ALTERNATE EXTERNAL MEMORY CONNECTION



© 1993 Microchip Technology Inc.

APPENDIX A: FFT ALGORITHM

```
MPASM B0.54                                                    PAGE  1

;*****************************************************************************
;                A Cooley-Tukey Radix-2 DIF FFT
;
;       Radix-2 implementation
;       Decimation In Frequency
;       Single Butterfly
;       Table Lookup of Twiddle Factors
;       Complex Input & Complex Output
;
;       All data is assumed to be 16 bits and the intermediate
;       results are stored in 32 bits
;
;       Length Of FFT must be a Power Of 2
;       Max Length Possible is 2**15
;
;       The input/output complex data is organized as a single array
;       of real data followed by imaginary data
;                Data is stored in External Memory and is accessed by
;       TABLRD & TABLWT Instructions
;
;*****************************************************************************
;
        LIST    P=17C42, C=120, T=ON, R=DEC, N=0
        include "17c42.h"
;
        include "17c42.mac"
;*****************************************************************************
;       RLC16AB
;
; DESCRIPTION:
;       16 bit rotate left A into B
;
; ARGUMENTS:
;       2*a => b
;
; TIMING (in cycles):
;       3
;
;
RLC16AB MACRO   a,b

        BCF     _carry
        RLCF    a+B0,W
        MOVWF   b+B0
        RLCF    a+B1,W
        MOVWF   b+B1
        ENDM
;*****************************************************************************
;       TBLADDR
;
; DESCRIPTION:
;       Load 16 bit table pointer with specified label
;
; TIMING (in cycles):
;       4
;
;
TBLADDR MACRO   label

        MOVLW   (label) & 0xff
        MOVWF   tblptrl
        MOVLW   page    (label)
        MOVWF   tblptrh
        ENDM

;*****************************************************************************
```


© 1993 Microchip Technology Inc.

```
                    ;       ADDLBL
                    ;
                    ; DESCRIPTION:
                    ;       Add A Label (16 bit constant) To A File Register (16 bit)
                    ;
                    ; TIMING (in cycles):
                    ;       4
                    ;
                    ADDLBL  MACRO   label,f
                            MOVLW   (label) & 0xff
                            ADDWF   f+B0
                            MOVLW   page    (label)
                            ADDWFC  f+B1
                            ENDM
                    ;*****************************************************************************
                    ;
0100                FftLen  .set    256                     ; FFT Length
0008                Power   .set    8                       ; (2**Power = FftLen)
00EF                DigitRevCount   .set    239             ; (FftLen-1) - (2**((Power+1)/2))

0001                SCALE_BUTTERFLY .set    TRUE            ; intermediate scaling performed

0800                EXT_RAM_START_ADDR      .set    0x0800  ; External Memory Data Storage Start Addr
                    ;
                    ;*****************************************************************************
                    ;
                            CBLOCK  0
0000 0004                           B0,B1,B2,B3             ; RAM offset constants
                            ENDC
                    ;
                            CBLOCK  0x18
0018 0002                           AARG,AARG1              ; 16 bit multiplier A
001A 0002                           BARG,BARG1              ; 16 bit multiplicand B
001C 0004                           DPX,DPX1,DPX2,DPX3      ; 32 bit multiplier result = A*B
                             ENDC
                    ;
                            CBLOCK
0020 0004                           ACC, ACC1, ACC2, ACC3   ; 32 bit accumulator for computa
                             ENDC
                    ;
                            CBLOCK
0024 0002                           count1,count11          ; N1
0026 0002                           count2,count22          ; N2
0028 0002                           QuartLen,QuartLen1      ; FftLen/4
                             ENDC
                    ;
                            CBLOCK
002A 0002                           TF_Offset,TF_Offset1
002C 0002                           TF_Addr,TF_Addr1        ; twiddle factor address computa
002E 0002                           Cos,Cos1,
0030 0002                           Sin,Sin1
                             ENDC
                    ;
                            CBLOCK
0032 0002                           VarIloop,VarIloop1
0034 0002                           VarJloop,VarJloop1
0036 0001                           VarKloop
0037 0002                           VarL,VarL1
                             ENDC
                    ;
                            CBLOCK
0039 0002                           Xi,Xi1
003B 0002                           Yi,Yi1
003D 0002                           Xl,Xl1
003F 0002                           Yl,Yl1
                             ENDC
                    ;
```


© 1993 Microchip Technology Inc.

```
                        CBLOCK
0041 0002                    Xt,Xt1
0043 0002                    Yt,Yt1
                        ENDC
                        CBLOCK
0045 0004                    temp,temp1,temp2,temp3
0049 0002                    testCount,testCount1
004B 0002                    PulseCount, PulseCount1
                        ENDC
                    ;
                    ;
                    ;*****************************************************************
                    ;               Test Program For FFT Subroutine
                    ;*****************************************************************
                    ;
                        ORG     0x0000
                    ;
                        include "square.asm"            ; Generate Test Vector Data
                    ;*****************************************************************
                    ;               Test Routine For FFT
                    ; FFT Of Square Wave Pulse
                    ;*****************************************************************
0008                PulseWidthFactor        .set    8
                    ;
                    testFft
                            MOVK16  2*PulseWidthFactor,testCount
0000 B010                    MOVLW   (2*PulseWidthFactor) & 0xff
0001 0149                    MOVWF   testCount+B0
0002 B000                    MOVLW   (2*PulseWidthFactor)/256
0003 014A                    MOVWF   testCount+B1
                             CLR16    Yi
0004 293B                    CLRF    Yi+B0
0005 293C                    CLRF    Yi+B1
                             TBLADDR ExtRamAddr         ; load table pointers with data start addr
0006 B000                    MOVLW   (ExtRamAddr) & 0xff
0007 010D                    MOVWF   tblptrl
0008 B008                    MOVLW   page    (ExtRamAddr)
0009 010E                    MOVWF   tblptrh
                    ;
                    nextPulse
                             MOVK    FftLen/PulseWidthFactor,PulseCount
000A B020                    MOVLW   FftLen/PulseWidthFactor
000B 014B                    MOVWF   PulseCount
                             MOVK    FftLen/PulseWidthFactor,PulseCount1
000C B020                    MOVLW   FftLen/PulseWidthFactor
000D 014C                    MOVWF   PulseCount1
                    ;
                            MOVK16  0x3FFF,Xi
000E B0FF                    MOVLW   (0x3FFF) & 0xff
000F 0139                    MOVWF   Xi+B0
0010 B03F                    MOVLW   (0x3FFF)/256
0011 013A                    MOVWF   Xi+B1
                             LX1
0012 E034                    call    write
0013 174B                    decfsz  PulseCount
0014 C012                    goto    LX1
                    ;
                            CLR16   Xi
0015 2939                    CLRF    Xi+B0
0016 293A                    CLRF    Xi+B1
                             LX2
0017 E034                    call    write
0018 174C                    decfsz  PulseCount1
0019 C017                    goto    LX2
                    ;
                            DEC16   testCount
```


© 1993 Microchip Technology Inc.

```
001A 2900                       CLRF    WREG
001B 0749                       DECF    testCount+B0
001C 034A                       SUBWFB  testCount+B1
                                TFSZ16  testCount
001D 6049                       MOVFP   testCount+B0,WREG
001E 084A                       IORWF   testCount+B1,W
001F 3300                       TSTFSZ  WREG
0020 C00A                       goto    nextPulse
                        ;
0021 E039                       call    R2FFT           ; Compute Fourier Transform
0022 E116                       call    Unscramble      ; Digit Reverse the scrambled data
                        ;
                        ; Fourier Transform Completed
                        ;
                        ; capture data to PIC-MASTER Trace Buffer
                                MOVK16  FftLen*2,temp
0023 B000                       MOVLW   (FftLen*2) & 0xff
0024 0145                       MOVWF   temp+B0
0025 B002                       MOVLW   (FftLen*2)/256
0026 0146                       MOVWF   temp+B1
                                TBLADDR ExtRamAddr      ; load table pointers with data start addr
0027 B000                       MOVLW   (ExtRamAddr) & 0xff
0028 010D                       MOVWF   tblptrl
0029 B008                       MOVLW   page    (ExtRamAddr)
002A 010E                       MOVWF   tblptrh
                                capture
002B A930                       tablrd  0,1,Sin         ; table latch = mem(tblptr)
                                DEC16   temp
002C 2900                       CLRF    WREG
002D 0745                       DECF    temp+B0
002E 0346                       SUBWFB  temp+B1
                                TFSZ16  temp
002F 6045                       MOVFP   temp+B0,WREG
0030 0846                       IORWF   temp+B1,W
0031 3300                       TSTFSZ  WREG
0032 C02B                       goto    capture
                        ;
0033 C033               self    goto    self
                        ;
                        write
0034 A439                       tlwt    0,Xi
0035 AF3A                       tablwt  1,1,Xi+B1       ; auto increment for Imag Data
0036 A43B                       tlwt    0,Yi
0037 AF3C                       tablwt  1,1,Yi+B1
0038 0002                       return
                        ;
                        ;*****************************************************************
                        ;                       RADIX-2 FFT
                        ;
                        ;       Decimation In Frequency
                        ;
                        ; Input Data should be unscrambled
                        ; Output Data at the end is in scrambled form
                        ; To obtain the unscrambled form, the digit reverse counter
                        ; subroutine, "Unscramble" should be called (see the example)
                        ;
                        ;*****************************************************************
                        R2FFT
                                MOVK16  FftLen,count2   ; count2 = N
0039 B000                       MOVLW   (FftLen) & 0xff
003A 0126                       MOVWF   count2+B0
003B B001                       MOVLW   (FftLen)/256
003C 0127                       MOVWF   count2+B1
                                MOVK16  FftLen/4,QuartLen ; QuartLen = FftLen/4
003D B040                       MOVLW   (FftLen/4) & 0xff
003E 0128                       MOVWF   QuartLen+B0
003F B000                       MOVLW   (FftLen/4)/256
0040 0129                       MOVWF   QuartLen+B1
```


© 1993 Microchip Technology Inc.

```
0041 292B                                          clrf    TF_Offset+B1
                        MOVK    1,TF_Offset        ; Init TF_Offset = 1
0042 B001               MOVLW   1
0043 012A               MOVWF   TF_Offset
                        MOVK    Power,VarKloop     ; Kloop
0044 B008               MOVLW   Power
0045 0136               MOVWF   VarKloop
                        Kloop                      ; for K = 1 to Power-1
                        MOV16   count2,count1      ; count1 = count2
0046 6026               MOVFP   count2+B0,WREG     ; get byte of count2 into w
0047 0124               MOVWF   count1+B0          ; move to count1(B0)
0048 6027               MOVFP   count2+B1,WREG     ; get byte of count2 into w
0049 0125               MOVWF   count1+B1          ; move to count1(B1)
                        RRC16   count2             ; count2 = count2/2
004A 1A27               RLCF    count2+B1,W        ; move sign into carry bit
004B 1927               RRCF    count2+B1
004C 1926               RRCF    count2+B0
                        CLR16   VarJloop           ; J = 0
004D 2934               CLRF    VarJloop+B0
004E 2935               CLRF    VarJloop+B1
                        CLR16   TF_Addr            ; TF_Addr = 0
004F 292C               CLRF    TF_Addr+B0
0050 292D               CLRF    TF_Addr+B1
                        Jloop
                ;
                ; Read Twiddle factors from Sine/Cosine Table from Prog Mem
                ;
                        MOVFP16 TF_Addr,tblptrl    ; load sine table address to table

0051 6D2C               MOVFP   TF_Addr+B0,tblptrl+B0 ; move TF_Addr(B0) to tblptrl(B0)
0052 6E2D               MOVFP   TF_Addr+B1,tblptrl+B1 ; move TF_Addr(B1) to tblptrl(B1)
                        ADDLBL  SineTable,tblptrl  ; add table offset
0053 B094               MOVLW   (SineTable) & 0xff
0054 0F0D               ADDWF   tblptrl+B0
0055 B002               MOVLW   page(SineTable)
0056 110E               ADDWFC  tblptrl+B1
0057 A830               tablrd  0,0,Sin            ; Read Sine Value from lookup table
0058 A030               tlrd    0,Sin
0059 A231               tlrd    1,Sin+B1
                        ADD16   QuartLen,tblptrl
005A 6028               MOVFP   QuartLen+B0,WREG   ; get lowest byte of QuartLen into w
005B 0F0D               ADDWF   tblptrl+B0         ; add lowest byte of tblptrl, save in
005C 6029               MOVFP   QuartLen+B1,WREG   ; get 2nd byte of QuartLen into w
005D 110E               ADDWFC  tblptrl+B1         ; add 2nd byte of tblptrl, save in
005E A82E               tablrd  0,0,Cos            ; Read Cosine Value from table
005F A02E               tlrd    0,Cos
0060 A22F               tlrd    1,Cos+B1
                ;
                        ADD16   TF_Offset,TF_Addr  ; TF_Addr = TF_Addr + TF_Offset
0061 602A               MOVFP   TF_Offset+B0,WREG  ; get lowest byte of TF_Offset into w
0062 0F2C               ADDWF   TF_Addr+B0         ; add lowest byte of TF_Addr, save in
0063 602B               MOVFP   TF_Offset+B1,WREG  ; get 2nd byte of TF_Offset into w
0064 112D               ADDWFC  TF_Addr+B1         ; add 2nd byte of TF_Addr, save in
            ;
                        RLC16AB VarJloop,VarIloop
0065 8804               BCF     _carry
0066 1A34               RLCF    VarJloop+B0,W
0067 0132               MOVWF   VarIloop+B0
0068 1A35               RLCF    VarJloop+B1,W
0069 0133               MOVWF   VarIloop+B1
            ;
                        Iloop
                        RLC16AB count2,VarL        ; VarL = count2*2
006A 8804               BCF     _carry
006B 1A26               RLCF    count2+B0,W
006C 0137               MOVWF   VarL+B0
006D 1A27               RLCF    count2+B1,W
006E 0138               MOVWF   VarL+B1
                        ADD16   VarIloop,VarL      ; VarL = (I+count2)*2
```


© 1993 Microchip Technology Inc.

```
006F 6032                   MOVFP   VarIloop+B0,WREG     ; get lowest byte of VarIloop into w
0070 0F37                   ADDWF   VarL+B0              ; add lowest byte of VarL, save in
0071 6033                   MOVFP   VarIloop+B1,WREG     ; get 2nd byte of VarIloop into w
0072 1138                   ADDWFC  VarL+B1              ; add 2nd byte of VarL, save in VarL(B1)
                    ;
                    ; Get Real & Imag Data from external RAMs (Program Memory)
                    ; load table pointers with data start addr
                    ;
                     MOVFP16 VarL,tblptrl             ; read data(L)
0073 6D37                   MOVFP   VarL+B0,tblptrl+B0   ; move VarL(B0) to tblptrl(B0)
0074 6E38                   MOVFP   VarL+B1,tblptrl+B1   ; move VarL(B1) to tblptrl(B1)
                            ADDLBL  ExtRamAddr,tblptrl   ; add data addr offset
0075 B000                   MOVLW   (ExtRamAddr) & 0xff
0076 0F0D                   ADDWF   tblptrl+B0
0077 B008                   MOVLW   page    (ExtRamAddr)
0078 110E                   ADDWFC  tblptrl+B1
0079 A93D                   tablrd  0,1,Xl               ; auto increment for Imag Data
007A A03D                   tlrd    0,Xl
007B A23E                   tlrd    1,Xl+B1              ; real data XL
007C A83F                   tablrd  0,0,Yl
007D A03F                   tlrd    0,Yl
007E A240                   tlrd    1,Yl+B1              ; imag data YL
                     MOVFP16 VarIloop,tblptrl         ; read data(I)
007F 6D32                   MOVFP   VarIloop+B0,tblptrl+B0; move VarIloop(B0) to tblptrl(B0)
0080 6E33                   MOVFP   VarIloop+B1,tblptrl+B1  move VarIloop(B1) to tblptrl(B1)
                            ADDLBL  ExtRamAddr,tblptrl   ; add data addr offset
0081 B000                   MOVLW   (ExtRamAddr) & 0xff
0082 0F0D                   ADDWF   tblptrl+B0
0083 B008                   MOVLW   page    (ExtRamAddr)
0084 110E                   ADDWFC  tblptrl+B1
0085 A939                   tablrd  0,1,Xi               ; auto increment for Imag Data
0086 A039                   tlrd    0,Xi
0087 A23A                   tlrd    1,Xi+B1              ; real data XI
0088 A83B                   tablrd  0,0,Yi
0089 A03B                   tlrd    0,Yi
008A A23C                   tlrd    1,Yi+B1              ; imag data YI
                    ;
                    ; Real & Imag Data is fetched
                    ; Compute Butterfly
                    ;
                    SUB16ACC        Xl,Xi,Xt         ; Xt = Xi - Xl
008B 603D                   MOVFP   Xl+B0,WREG           ; get lowest byte of Xl into w
008C 0439                   SUBWF   Xi+B0,W              ; sub lowest byte of Xi, save in Xi(B0)
008D 0141                   MOVWF   Xt+B0
008E 603E                   MOVFP   Xl+B1,WREG           ; get 2nd byte of Xl into w
008F 023A                   SUBWFB  Xi+B1,W              ; sub 2nd byte of Xi, save in Xi(B1)
0090 0142                   MOVWF   Xt+B1
                            ADD16   Xl,Xi                ; Xi = Xi + Xl
0091 603D                   MOVFP   Xl+B0,WREG           ; get lowest byte of Xl into w
0092 0F39                   ADDWF   Xi+B0                ; add lowest byte of Xi, save in Xi(B0)
0093 603E                   MOVFP   Xl+B1,WREG           ; get 2nd byte of Xl into w
0094 113A                   ADDWFC  Xi+B1                ; add 2nd byte of Xi, save in Xi(B1)
                            SUB16ACC        Yl,Yi,Yt     ; Yt = Yi - Yl
0095 603F                   MOVFP   Yl+B0,WREG           ; get lowest byte of Yl into w
0096 043B                   SUBWF   Yi+B0,W              ; sub lowest byte of Yi, save in Yi(B0)
0097 0143                   MOVWF   Yt+B0
0098 6040                   MOVFP   Yl+B1,WREG           ; get 2nd byte of Yl into w
0099 023C                   SUBWFB  Yi+B1,W              ; sub 2nd byte of Yi, save in Yi(B1)
009A 0144                   MOVWF   Yt+B1
                            ADD16           Yl,Yi        ; Yi = Yi + Yl
009B 603F                   MOVFP   Yl+B0,WREG           ; get lowest byte of Yl into w
009C 0F3B                   ADDWF   Yi+B0                ; add lowest byte of Yi, save in Yi(B0)
009D 6040                   MOVFP   Yl+B1,WREG           ; get 2nd byte of Yl into w
009E 113C                   ADDWFC  Yi+B1                ; add 2nd byte of Yi, save in Yi(B1)
                    ;
                    if SCALE_BUTTERFLY
                    RRC16   Xi
009F 1A3A                   RLCF    Xi+B1,W          ; move sign into carry bit
00A0 193A                   RRCF    Xi+B1
```

4

```
00A1 1939                       RRCF    Xi+B0
                                RRC16   Yi
00A2 1A3C                       RLCF    Yi+B1,W             ; move sign into carry bit
00A3 193C                       RRCF    Yi+B1
00A4 193B                       RRCF    Yi+B0
                                RRC16   Xt
00A5 1A42                       RLCF    Xt+B1,W             ; move sign into carry bit
00A6 1942                       RRCF    Xt+B1
00A7 1941                       RRCF    Xt+B0
                                RRC16   Yt
00A8 1A44                       RLCF    Yt+B1,W             ; move sign into carry bit
00A9 1944                       RRCF    Yt+B1
00AA 1943                       RRCF    Yt+B0
                                endif
                    ;

                                MOVFP16 Cos,AARG
00AB 782E                       MOVFP   Cos+B0,AARG+B0      ; move Cos(B0) to AARG(B0)
00AC 792F                       MOVFP   Cos+B1,AARG+B1      ; move Cos(B1) to AARG(B1)
                                MOVFP16 Yt,BARG
00AD 7A43                       MOVFP   Yt+B0,BARG+B0       ; move Yt(B0) to BARG(B0)
00AE 7B44                       MOVFP   Yt+B1,BARG+B1       ; move Yt(B1) to BARG(B1)
00AF E182                       Call    DblMult ; COS*Yt
                                MOVPF32 DPX,ACC
00B0 5C20                       MOVPF   DPX+B0,ACC+B0       ; move DPX(B0) to ACC(B0)
00B1 5D21                       MOVPF   DPX+B1,ACC+B1       ; move DPX(B1) to ACC(B1)
00B2 5E22                       MOVPF   DPX+B2,ACC+B2       ; move DPX(B2) to ACC(B2)
00B3 5F23                       MOVPF   DPX+B3,ACC+B3       ; move DPX(B3) to ACC(B3)
                                MOVFP16 Sin,AARG
00B4 7830                       MOVFP   Sin+B0,AARG+B0      ; move Sin(B0) to AARG(B0)
00B5 7931                       MOVFP   Sin+B1,AARG+B1      ; move Sin(B1) to AARG(B1)
                                MOVFP16 Xt,BARG
00B6 7A41                       MOVFP   Xt+B0,BARG+B0       ; move Xt(B0) to BARG(B0)
00B7 7B42                       MOVFP   Xt+B1,BARG+B1       ; move Xt(B1) to BARG(B1)
00B8 E182                       Call    DblMult ; SIN*Xt, Scale if necessary
                                ADD32   ACC,DPX
00B9 6020                       MOVFP   ACC+B0,WREG         ; get lowest byte of ACC into w
00BA 0F1C                       ADDWF   DPX+B0              ; add lowest byte of DPX, save in DPX(B0)
00BB 6021                       MOVFP   ACC+B1,WREG         ; get 2nd byte of ACC into w
00BC 111D                       ADDWFC  DPX+B1              ; add 2nd byte of DPX, save in DPX(B1)
00BD 6022                       MOVFP   ACC+B2,WREG         ; get 3rd byte of ACC into w
00BE 111E                       ADDWFC  DPX+B2              ; add 3rd byte of DPX, save in DPX(B2)
00BF 6023                       MOVFP   ACC+B3,WREG         ; get 4th byte of ACC into w
00C0 111F                       ADDWFC  DPX+B3              ; add 4th byte of DPX, save in DPX(B3)
                                MOVPF16 DPX+B2,Yl           ; Yl = COS*Yt + SIN*Xt, Scale if neces
00C1 5E3F                       MOVPF   DPX+B2+B0,Yl+B0     ; move DPX+B2(B0) to Yl(B0)
00C2 5F40                       MOVPF   DPX+B2+B1,Yl+B1     ; move DPX+B2(B1) to Yl(B1)
                    ;
                                MOVFP16 Yt,BARG             ; AARG = SIN, BARG = Yt
00C3 7A43                       MOVFP   Yt+B0,BARG+B0       ; move Yt(B0) to BARG(B0)
00C4 7B44                       MOVFP   Yt+B1,BARG+B1       ; move Yt(B1) to BARG(B1)
00C5 E182                       Call    DblMult ; SIN*Yt
                                MOVPF32 DPX,ACC
00C6 5C20                       MOVPF   DPX+B0,ACC+B0       ; move DPX(B0) to ACC(B0)
00C7 5D21                       MOVPF   DPX+B1,ACC+B1       ; move DPX(B1) to ACC(B1)
00C8 5E22                       MOVPF   DPX+B2,ACC+B2       ; move DPX(B2) to ACC(B2)
00C9 5F23                       MOVPF   DPX+B3,ACC+B3       ; move DPX(B3) to ACC(B3)
                                MOVFP16 Cos,AARG
00CA 782E                       MOVFP   Cos+B0,AARG+B0      ; move Cos(B0) to AARG(B0)
00CB 792F                       MOVFP   Cos+B1,AARG+B1      ; move Cos(B1) to AARG(B1)
                                MOVFP16 Xt,BARG
00CC 7A41                       MOVFP   Xt+B0,BARG+B0       ; move Xt(B0) to BARG(B0)
00CD 7B42                       MOVFP   Xt+B1,BARG+B1       ; move Xt(B1) to BARG(B1)
00CE E182                       Call    DblMult             ; COS*Xt, Scale if necessary
                                SUB32   ACC,DPX             ; DPX = COS*Xt - SIN*Yt
00CF 6020                       MOVFP   ACC+B0,WREG         ; get lowest byte of ACC into w
00D0 051C                       SUBWF   DPX+B0              ; sub lowest byte of DPX, save in DPX(B0)
00D1 6021                       MOVFP   ACC+B1,WREG         ; get 2nd byte of ACC into w
00D2 031D                       SUBWFB  DPX+B1              ; sub 2nd byte of DPX, save in DPX(B1)
```


© 1993 Microchip Technology Inc.

```
00D3 6022                MOVFP   ACC+B2,WREG         ; get 3rd byte of ACC into w
00D4 031E                SUBWFB  DPX+B2              ; sub 3rd byte of DPX, save in DPX(B2)
00D5 6023                MOVFP   ACC+B3,WREG         ; get 4th byte of ACC into w
00D6 031F                SUBWFB  DPX+B3              ; sub 4th byte of DPX, save in DPX(B3)
                         MOVPF16 DPX+B2,Xl           ; Xl = COS*Xt - SIN*Yt, Scale if neces
00D7 5E3D                MOVPF   DPX+B2+B0,Xl+B0     ; move DPX+B2(B0) to Xl(B0)
00D8 5F3E                MOVPF   DPX+B2+B1,Xl+B1     ; move DPX+B2(B1) to Xl(B1)
                    ;
                    ;
                    ; Store results of butterfly
                    ;
                         DEC16   tblptrl             ; table pointer already loaded with I
00D9 2900                CLRF    WREG
00DA 070D                DECF    tblptrl+B0
00DB 030E                SUBWFB  tblptrl+B1
00DC A439                tlwt    0,Xi
00DD AF3A                tablwt  1,1,Xi+B1           ; auto increment for Imag Data
00DE A43B                tlwt    0,Yi
00DF AE3C                tablwt  1,0,Yi+B1           ; Xi & Yi stored
                         MOVFP16 VarL,tblptrl        ; read data(L)
00E0 6D37                MOVFP   VarL+B0,tblptrl+B   ; move VarL(B0) to tblptrl(B0)
00E1 6E38                MOVFP   VarL+B1,tblptrl+B1  ; move VarL(B1) to tblptrl(B1)
                         ADDLBL  ExtRamAddr,tblptrl  ; add data addr offset
00E2 B000                MOVLW   (ExtRamAddr) & 0xff
00E3 0F0D                ADDWF   tblptrl+B0
00E4 B008                MOVLW   page    (ExtRamAddr)
00E5 110E                ADDWFC  tblptrl+B1
00E6 A43D                tlwt    0,Xl
00E7 AF3E                tablwt  1,1,Xl+B1           ; auto increment for Imag Data
00E8 A43F                tlwt    0,Yl
00E9 AE40                tablwt  1,0,Yl+B1           ; X(L) & Y(L) stored
                    ;
                    ; Increment for next Iloop
                    ;
                         RLC16AB count1,temp         ; temp = count1*2
00EA 8804                BCF     _carry
00EB 1A24                RLCF    count1+B0,W
00EC 0145                MOVWF   temp+B0
00ED 1A25                RLCF    count1+B1,W
00EE 0146                MOVWF   temp+B1
                         ADD16   temp,VarIloop       ; I = I + temp
00EF 6045                MOVFP   temp+B0,WREG        ; get lowest byte of temp into w
00F0 0F32                ADDWF   VarIloop+B0         ; add lowest byte of VarIloop,save in VarIloop(B
00F1 6046                MOVFP   temp+B1,WREG        ; get 2nd byte of temp into w
00F2 1133                ADDWFC  VarIloop+B1         ; add 2nd byte of VarIloop,save in VarIloop(B1)
                    ;
                         MOVK16  (FftLen*2),temp
00F3 B000                MOVLW   ((FftLen*2)) & 0xff
00F4 0145                MOVWF   temp+B0
00F5 B002                MOVLW   ((FftLen*2))/256
00F6 0146                MOVWF   temp+B1
                         SUB16   VarIloop,temp       ; temp = 2*FftLen - I
00F7 6032                MOVFP   VarIloop+B0,WREG    ; get lowest byte of VarIloop into w
00F8 0545                SUBWF   temp+B0             ; sub lowest byte of temp, save in temp(B0)
00F9 6033                MOVFP   VarIloop+B1,WREG    ; get 2nd byte of VarIloop into w
00FA 0346                SUBWFB  temp+B1             ; sub 2nd byte of temp, save in temp(B1)
                         DEC16   temp
00FB 2900                CLRF    WREG
00FC 0745                DECF    temp+B0
00FD 0346                SUBWFB  temp+B1
00FE 9746                btfss   temp+B1,MSB
00FF C06A                goto    Iloop               ; while I < 2*FftLen
                    ;
                    ; I Loop end
```

4

© 1993 Microchip Technology Inc.

```
                        ;
                        ; increment for next J Loop
                        ;
                        INC16   VarJloop            ; J = J + 1
0100 2900               CLRF    WREG
0101 1534               INCF    VarJloop+B0
0102 1135               ADDWFC  VarJloop+B1
                        MOV16   count2,temp
0103 6026               MOVFP   count2+B0,WREG      ; get byte of count2 into w
0104 0145               MOVWF   temp+B0             ; move to temp(B0)
0105 6027               MOVFP   count2+B1,WREG      ; get byte of count2 into w
0106 0146               MOVWF   temp+B1             ; move to temp(B1)
                        SUB16   VarJloop,temp       ; temp = count2 - J
0107 6034               MOVFP   VarJloop+B0,WREG    ; get lowest byte of VarJloop into w
0108 0545               SUBWF   temp+B0             ; sub lowest byte of temp, save in temp(B0)
0109 6035               MOVFP   VarJloop+B1,WREG    ; get 2nd byte of VarJloop into w
010A 0346               SUBWFB  temp+B1             ; sub 2nd byte of temp, save in temp(B1)
                        DEC16   temp
010B 2900               CLRF    WREG
010C 0745               DECF    temp+B0
010D 0346               SUBWFB  temp+B1
010E 9746               btfss   temp+B1,MSB
010F C051               goto    Jloop               ; while J < count2
                        ;
                        ; J Loop end
                        ;
                        ; increment for next K Loop
                        ;
                        RLC16   TF_Offset           ; TF_Offset = 2 * TF_Offset
0110 8804               BCF     _carry
0111 1B2A               RLCF    TF_Offset+B0
0112 1B2B               RLCF    TF_Offset+B1
0113 1736               decfsz  VarKloop
0114 C046               goto    Kloop               ; while K < Power
                        ;
0115 0002               return                      ; FFT complete
                        ;
                        ; K Loop End
                        ; FFT Computation Over with data scrambled
                        ; Descramble the data using "Unscramble" Routine
                        ;
                        ;*****************************************************************
                        ;       Unscramble Data Order Sequence
                        ;       A digit reverse counter
                        ;*****************************************************************
                        include "reverse.asm"
                        ;*****************************************************************
                        ;                       A digit reverse counter
                        ;
                        ;       Unscramble Data Order Sequence Of Radix-2 FFT
                        ;       Length (must be a power of 2) is limited only by
                        ;       the amount of External RAM available and must be
                        ;       a number less than 2**15
                        ;
                        ;*****************************************************************
                        Unscramble
                                CLR16   VarJloop            ; J = 0
0116 2934               CLRF    VarJloop+B0
0117 2935               CLRF    VarJloop+B1
0118 2933               clrf    VarIloop+B1
                        MOVK    1,VarIloop          ; I = 1
0119 B001               MOVLW   1
011A 0132               MOVWF   VarIloop
                nextI
                        MOVK16  FftLen/2,VarKloop
011B B080               MOVLW   (FftLen/2) & 0xff
011C 0136               MOVWF   VarKloop+B0
011D B000               MOVLW   (FftLen/2)/256
```


© 1993 Microchip Technology Inc.

```
011E 0137                       MOVWF   VarKloop+B1
011F C127                       goto    testK
                        KlessJ
                                SUB16   VarKloop,VarJloop  ; J = J - K
0120 6036                       MOVFP   VarKloop+B0,WREG   ; get lowest byte of VarKloop into w
0121 0534                       SUBWF   VarJloop+B0        ; sub lowest byte of VarJloop,save in VarJloop(B0)
0122 6037                       MOVFP   VarKloop+B1,WREG   ; get 2nd byte of VarKloop into w
0123 0335                       SUBWFB  VarJloop+B1        ; sub 2nd byte of VarJloop,save in VarJloop(B1)
                                RRC16   VarKloop           ; K = K/2
0124 1A37                       RLCF    VarKloop+B1,W      ; move sign into carry bit
0125 1937                       RRCF    VarKloop+B1
0126 1936                       RRCF    VarKloop+B0
                        testK
                                MOV16   VarJloop,temp

0127 6034                       MOVFP   VarJloop+B0,WREG   ; get byte of VarJloop into w
0128 0145                       MOVWF   temp+B0            ; move to temp(B0)
0129 6035                       MOVFP   VarJloop+B1,WREG   ; get byte of VarJloop into w
012A 0146                       MOVWF   temp+B1            ; move to temp(B1)
                                SUB16   VarKloop,temp      ; temp = J - K
012B 6036                       MOVFP   VarKloop+B0,WREG   ; get lowest byte of VarKloop into w
012C 0545                       SUBWF   temp+B0            ; sub lowest byte of temp, save in temp(B0)
012D 6037                       MOVFP   VarKloop+B1,WREG   ; get 2nd byte of VarKloop into w
012E 0346                       SUBWFB  temp+B1            ; sub 2nd byte of temp, save in temp(B1)
012F 9746                       btfss   temp+B1,MSB
0130 C120                       goto    KlessJ             ; while K < J
                                ADD16   VarKloop,VarJloop  ; J = J + K
0131 6036                       MOVFP   VarKloop+B0,WREG   ; get lowest byte of VarKloop into w
0132 0F34                       ADDWF   VarJloop+B0        ; add lowest byte of VarJloop,save in VarJloop(B0)
0133 6037                       MOVFP   VarKloop+B1,WREG   ; get 2nd byte of VarKloop into w
0134 1135                       ADDWFC  VarJloop+B1        ; add 2nd byte of VarJloop,save in VarJloop(B1)
                        ;
                        ; if (i < j) then swap data(i) & data(j)
                        ;
                                MOV16   VarJloop,temp
0135 6034                       MOVFP   VarJloop+B0,WREG   ; get byte of VarJloop into w
0136 0145                       MOVWF   temp+B0            ; move to temp(B0)
0137 6035                       MOVFP   VarJloop+B1,WREG   ; get byte of VarJloop into w
0138 0146                       MOVWF   temp+B1            ; move to temp(B1)
                                SUB16   VarIloop,temp      ; temp = J - I
0139 6032                       MOVFP   VarIloop+B0,WREG   ; get lowest byte of VarIloop into w
013A 0545                       SUBWF   temp+B0            ; sub lowest byte of temp, save in temp(B0)
013B 6033                       MOVFP   VarIloop+B1,WREG   ; get 2nd byte of VarIloop into w
013C 0346                       SUBWFB  temp+B1            ; sub 2nd byte of temp, save in temp(B1)
                                DEC16   temp
013D 2900                       CLRF    WREG
013E 0745                       DECF    temp+B0
013F 0346                       SUBWFB  temp+B1
0140 9F46                       btfsc   temp+B1,MSB
0141 C174                       goto    incI
                        ;
                        ; swap data
                        ; read data(i)
                                RLC16AB VarIloop,tblptrl   ; add twice the addr, since Real Data
0142 8804                       BCF     _carry
0143 1A32                       RLCF    VarIloop+B0,W
0144 010D                       MOVWF   tblptrl+B0
0145 1A33                       RLCF    VarIloop+B1,W
0146 010E                       MOVWF   tblptrl+B1
                                ADDLBL  ExtRamAddr,tblptrl ; is followed by Imag Data
0147 B000                       MOVLW   (ExtRamAddr) & 0xff
0148 0F0D                       ADDWF   tblptrl+B0
0149 B008                       MOVLW   page    (ExtRamAddr)
014A 110E                       ADDWFC  tblptrl+B1
014B A939                       tablrd  0,1,Xi    ; auto increment for Imag Data
014C A039                       tlrd    0,Xi
014D A23A                       tlrd    1,Xi+B1   ; real data XI
014E A83B                       tablrd  0,0,Yi
014F A03B                       tlrd    0,Yi
```

```
0150 A23C                tlrd    1,Yi+B1  ; imag data YI
                    ;
                    ; read data(j)
                    ;
                         RLC16AB VarJloop,tblptrl   ; add twice the addr, since Real Data
0151 8804                BCF     _carry
0152 1A34                RLCF    VarJloop+B0,W
0153 010D                MOVWF   tblptrl+B0
0154 1A35                RLCF    VarJloop+B1,W
0155 010E                MOVWF   tblptrl+B1
                         ADDLBL  ExtRamAddr,tblptrl ; is followed by Imag Data
0156 B000                MOVLW   (ExtRamAddr) & 0xff
0157 0F0D                ADDWF   tblptrl+B0
0158 B008                MOVLW   page    (ExtRamAddr)
0159 110E                ADDWFC  tblptrl+B1
015A A93D                tablrd  0,1,Xl  ; auto increment for Imag Data
015B A03D                tlrd    0,Xl
015C A23E                tlrd    1,Xl+B1 ; real data XL
015D A83F                tablrd  0,0,Yl
015E A03F                tlrd    0,Yl
015F A240                tlrd    1,Yl+B1 ; imag data YL
                    ;
                    ; Interchange data(I) & data(J)
                    ;
                    ; J addr already loaded into table pointers, bu autoincremented
                    ;
                         DEC16   tblptrl
0160 2900                CLRF    WREG
0161 070D                DECF    tblptrl+B0
0162 030E                SUBWFB  tblptrl+B1
0163 A439                tlwt    0,Xi
0164 AF3A                tablwt  1,1,Xi+B1          ; auto increment for Imag Data
0165 A43B                tlwt    0,Yi
0166 AE3C                tablwt  1,0,Yi+B1          ; X(I) & Y(I) stored
                         RLC16AB VarIloop,tblptrl   ; add twice the addr, since Real Data
0167 8804                BCF     _carry
0168 1A32                RLCF    VarIloop+B0,W
0169 010D                MOVWF   tblptrl+B0
016A 1A33                RLCF    VarIloop+B1,W
016B 010E                MOVWF   tblptrl+B1
                         ADDLBL  ExtRamAddr,tblptrl ; is followed by Imag Data
016C B000                MOVLW   (ExtRamAddr) & 0xff
016D 0F0D                ADDWF   tblptrl+B0
016E B008                MOVLW   page    (ExtRamAddr)
016F 110E                ADDWFC  tblptrl+B1
0170 A43D                tlwt    0,Xl
0171 AF3E                tablwt  1,1,Xl+B1          ; auto increment for Imag Data
0172 A43F                tlwt    0,Yl
0173 AE40                tablwt  1,0,Yl+B1          ; X(L) & Y(L) stored
                    ;
                    ; increment I
                    ;
                    incI
                         INC16   VarIloop
0174 2900                CLRF    WREG
0175 1532                INCF    VarIloop+B0
0176 1133                ADDWFC  VarIloop+B1
                         MOVK16  DigitRevCount,temp
0177 B0EF                MOVLW   (DigitRevCount) & 0xff
0178 0145                MOVWF   temp+B0
0179 B000                MOVLW   (DigitRevCount)/256
017A 0146                MOVWF   temp+B1
                         SUB16   VarIloop,temp      ; temp = DigitRevCount - I
017B 6032                MOVFP   VarIloop+B0,WREG   ; get lowest byte of VarIloop into w
017C 0545                SUBWF   temp+B0            ; sub lowest byte of temp, save in temp(B0)
017D 6033                MOVFP   VarIloop+B1,WREG   ; get 2nd byte of VarIloop into w
017E 0346                SUBWFB  temp+B1            ; sub 2nd byte of temp, save in temp(B1)
017F 9746                btfss   temp+B1,MSB
0180 C11B                goto    nextI              ; while i < DigitRevCount
```


© 1993 Microchip Technology Inc.

```
0181 0002                       return
                        ;
                        ; End digit reverse counter
                        ;
                        ;*************************************************************************


                        ;*************************************************************
                        ;       Include Double Precision Multiplication Routine
                        ;*************************************************************
0001                    SIGNED  equ     TRUE

                                include "17c42mpy.mac"


                        ;
                        ;*************************************************************
                        ;               Sine-Cosine Tables
                        ;*************************************************************
                        ;
                                include "fft256.tbl"
                        ;
                        ;       256 Point FFT Sine Table
                        ; coefficient table (size of table is 3n/4).
                        ;
                        SineTable
                        ;
0294 0000                               data            0
0295 0324                               data            804
0296 0648                               data            1608
0297 096A                               data            2410
0298 0C8C                               data            3212
0299 0FAB                               data            4011
029A 12C8                               data            4808
029B 15E2                               data            5602
029C 18F9                               data            6393
029D 1C0B                               data            7179
029E 1F1A                               data            7962
029F 2223                               data            8739
02A0 2528                               data            9512
02A1 2826                               data            10278
02A2 2B1F                               data            11039
02A3 2E11                               data            11793
02A4 30FB                               data            12539
02A5 33DF                               data            13279
02A6 36BA                               data            14010
02A7 398C                               data            14732
02A8 3C56                               data            15446
02A9 3F17                               data            16151
02AA 41CE                               data            16846
02AB 447A                               data            17530
02AC 471C                               data            18204
02AD 49B4                               data            18868
02AE 4C3F                               data            19519
02AF 4EBF                               data            20159
02B0 5133                               data            20787
02B1 539B                               data            21403
02B2 55F5                               data            22005
02B3 5842                               data            22594
02B4 5A82                               data            23170
02B5 5CB3                               data            23731
02B6 5ED7                               data            24279
02B7 60EB                               data            24811
02B8 62F1                               data            25329
02B9 64E8                               data            25832
02BA 66CF                               data            26319
02BB 68A6                               data            26790
02BC 6A6D                               data            27245
```

4

© 1993 Microchip Technology Inc.

```
02BD 6C23                   data            27683
02BE 6DC9                   data            28105
02BF 6F5E                   data            28510
02C0 70E2                   data            28898
02C1 7254                   data            29268
02C2 73B5                   data            29621
02C3 7504                   data            29956
02C4 7641                   data            30273
02C5 776B                   data            30571
02C6 7884                   data            30852
02C7 7989                   data            31113
02C8 7A7C                   data            31356
02C9 7B5C                   data            31580
02CA 7C29                   data            31785
02CB 7CE3                   data            31971
02CC 7D89                   data            32137
02CD 7E1D                   data            32285
02CE 7E9C                   data            32412
02CF 7F09                   data            32521
02D0 7F61                   data            32609
02D1 7FA6                   data            32678
02D2 7FD8                   data            32728
02D3 7FF5                   data            32757
                    ;
                    CosTable
                    ;
02D4 7FFF                   data            32767
02D5 7FF5                   data            32757
02D6 7FD8                   data            32728
02D7 7FA6                   data            32678
02D8 7F61                   data            32609
02D9 7F09                   data            32521
02DA 7E9C                   data            32412
02DB 7E1D                   data            32285
02DC 7D89                   data            32137
02DD 7CE3                   data            31971
02DE 7C29                   data            31785
02DF 7B5C                   data            31580
02E0 7A7C                   data            31356
02E1 7989                   data            31113
02E2 7884                   data            30852
02E3 776B                   data            30571
02E4 7641                   data            30273
02E5 7504                   data            29956
02E6 73B5                   data            29621
02E7 7254                   data            29268
02E8 70E2                   data            28898
02E9 6F5E                   data            28510
02EA 6DC9                   data            28105
02EB 6C23                   data            27683
02EC 6A6D                   data            27245
02ED 68A6                   data            26790
02EE 66CF                   data            26319
02EF 64E8                   data            25832
02F0 62F1                   data            25329
02F1 60EB                   data            24811
02F2 5ED7                   data            24279
02F3 5CB3                   data            23731
02F4 5A82                   data            23170
02F5 5842                   data            22594
02F6 55F5                   data            22005
02F7 539B                   data            21403
02F8 5133                   data            20787
02F9 4EBF                   data            20159
02FA 4C3F                   data            19519
02FB 49B4                   data            18868
02FC 471C                   data            18204
02FD 447A                   data            17530
02FE 41CE                   data            16846
```


© 1993 Microchip Technology Inc.

```
02FF 3F17                data            16151
0300 3C56                data            15446
0301 398C                data            14732
0302 36BA                data            14010
0303 33DF                data            13279
0304 30FB                data            12539
0305 2E11                data            11793
0306 2B1F                data            11039
0307 2826                data            10278
0308 2528                data            9512
0309 2223                data            8739
030A 1F1A                data            7962
030B 1C0B                data            7179
030C 18F9                data            6393
030D 15E2                data            5602
030E 12C8                data            4808
030F 0FAB                data            4011
0310 0C8C                data            3212
0311 096A                data            2410
0312 0648                data            1608
0313 0324                data            804
0314 0000                data            0
0315 FCDC                data            -804
0316 F9B8                data            -1608
0317 F696                data            -2410
0318 F374                data            -3212
0319 F055                data            -4011
031A ED38                data            -4808
031B EA1E                data            -5602
031C E707                data            -6393
031D E3F5                data            -7179
031E E0E6                data            -7962
031F DDDD                data            -8739
0320 DAD8                data            -9512
0321 D7DA                data            -10278
0322 D4E1                data            -11039
0323 D1EF                data            -11793
0324 CF05                data            -12539
0325 CC21                data            -13279
0326 C946                data            -14010
0327 C674                data            -14732
0328 C3AA                data            -15446
0329 C0E9                data            -16151
032A BE32                data            -16846
032B BB86                data            -17530
032C B8E4                data            -18204
032D B64C                data            -18868
032E B3C1                data            -19519
032F B141                data            -20159
0330 AECD                data            -20787
0331 AC65                data            -21403
0332 AA0B                data            -22005
0333 A7BE                data            -22594
0334 A57E                data            -23170
0335 A34D                data            -23731
0336 A129                data            -24279
0337 9F15                data            -24811
0338 9D0F                data            -25329
0339 9B18                data            -25832
033A 9931                data            -26319
033B 975A                data            -26790
033C 9593                data            -27245
033D 93DD                data            -27683
033E 9237                data            -28105
033F 90A2                data            -28510
0340 8F1E                data            -28898
0341 8DAC                data            -29268
0342 8C4B                data            -29621
0343 8AFC                data            -29956
```


© 1993 Microchip Technology Inc.

```
0344 89BF                       data            -30273
0345 8895                       data            -30571
0346 877C                       data            -30852
0347 8677                       data            -31113
0348 8584                       data            -31356
0349 84A4                       data            -31580
034A 83D7                       data            -31785
034B 831D                       data            -31971
034C 8277                       data            -32137
034D 81E3                       data            -32285
034E 8164                       data            -32412
034F 80F7                       data            -32521
0350 809F                       data            -32609
0351 805A                       data            -32678
0352 8028                       data            -32728
0353 800B                       data            -32757
                        ;
                        ;***********************************************


                        ;
                        ;*****************************************************************
                        ;       FFT Input/Output Data Stored In External RAM
                        ; Operate Processor In Extended Microcontroller Mode
                        ; External Data Starts at Address 0x0800, with 2 bytes of
                        ; Real Data followed by 2 bytes of Imaginary Data.
                        ;*****************************************************************
                                ORG     EXT_RAM_START_ADDR
                        ;
                        ExtRamAddr
                        ;
                                END
Errors   :     0
Warnings :     0
```


© 1993 Microchip Technology Inc.

MICROCHIP

AN543

Tone Generation

INTRODUCTION

A general purpose resonator routine is implemented using PIC17C42. This routine is used to generate multiple tones. A tone signal is normally generated using extensive table lookup schemes. When a multiple tone signal is desired, each tone must have its own lookup table, thus requiring a large amount of storage space, especially when various frequencies are to be generated. This application note implements a tone generation using recursive techniques. The algorithm for a resonator is developed and implemented using PIC17C42.

THEORY

Generation of a single tone basically implies generating samples of a sine/cosine wave. The Z-Transform of a sine wave is given as follows :

$$Z\{\sin(wt)\} = \frac{Y(z)}{X(z)} = \frac{z^*\sin(wT)}{z^2 - 2^*z^*\cos(wT) + 1}$$

The impulse response of the above transform (i.e. for X(z) = 1) will generate a sine wave of frequency w sampled at a rate of T (= 1/fs). Thus the above equation is translated to:

$$Y(z) = \frac{z^{-1}*\sin(wT)}{1 - 2^*z^{-1}*\cos(wT) + z^{-2}}$$

The above equation can be rewritten in a difference equation form as follows:

$$y(n) - 2y(n-1)\cos(wT) + y(n-2) = x(n-1)\sin(wT)$$

Rearranging the above equation and setting, x(n) as an impulse sequence, the following recursive equations are obtained.

$$y(n) = 2^*K_1^*y(n-1) - y(n-2)$$

$$y(n-2) = y(n-1)$$

$$y(n-1) = y(n)$$

with the following conditions:

$$K_1 = \cos(wT)$$

$$K_2 = \text{initial } y(n-1) = \sin(wT)$$

$$K_3 = \text{initial } y(n-2) = 0$$

IMPLEMENTATION

The above developed algorithm is implemented as a subroutine using PIC17C42. All computations are performed using double precision arithmetic (16/32-bits). The recursive tone generation algorithm is implemented as a subroutine (labelled as "Resonator"). This subroutine generates samples of a single tone. To generate multiple frequencies, simply call this resonator routine for the desired frequencies and sum the outputs. The three tone co-efficients are stored in program memory and are read into the data memory using `TABLRD` instructions.

The fully commented code is listed in Appendix A. The timing and memory requirements are included in the comment sections of the code. For a listing of the header file "17C42.h" and the macro definition file "17C42.mac" please refer to Appendices C and D respectively of the application note ANDS00540. This code can be easily modified and used in various applications like DTMF generation, sound generation, etc. The tones generated can easily be output to an on chip PWM channel which in turn can drive a speaker for producing various sounds. If using a PWM channel, it is suggested to set the PWM frequency much higher than the sampling frequency used (in the example code, for 8 KHz sampling frequency, use at least 20 KHz PWM frequency).

As an example, a dual tone is generated and the resulting digital wave form is analyzed. The main program calls the function "Resonator" twice for generating the two desired tones and the two outputs are summed. A sampling frequency of 8 KHz was used to generate a dual tone of 800 Hz and 1.10 KHz. The resulting wave form is shown in Figure 1. The spectrum of the signal shown in Figure 2 shows two peaks corresponding to the two desired tones (800 Hz & 1.10 KHz). The assembly code was tested using PICMASTER™ (Microchip's Universal In-Circuit Emulator System).

The generated tones were captured into the PICMASTER's trace buffer and then transferred to Microsoft Excel® using Dynamic Data Exchange (DDE). Once the data is in Excel™ it is analyzed using Excel's FFT utility.

4

© 1993 Microchip Technology Inc.

FIGURE 1 - DUAL TONE WAVE FORM CAPTURED BY PICMASTER

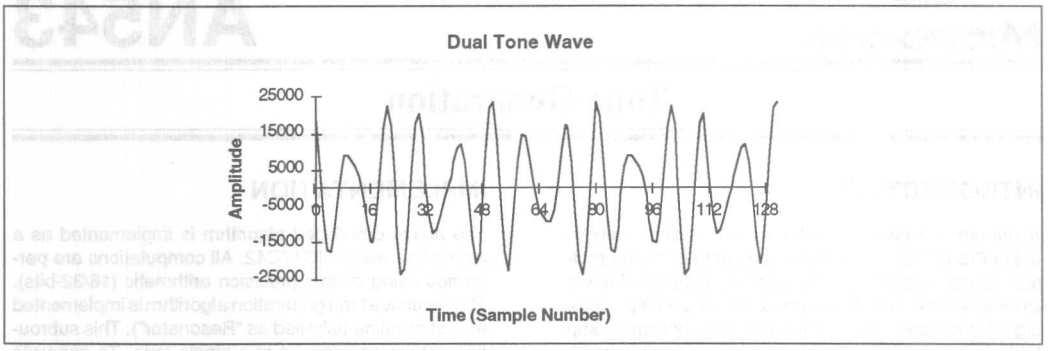
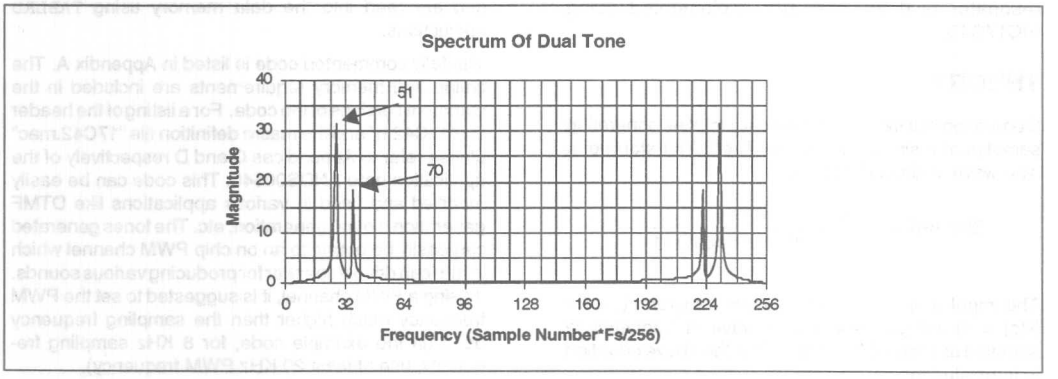


FIGURE 2 - FREQUENCY SPECTRUM OF FIG. 1.1 SHOWING THE DUAL TONE FREQUENCIES



PERFORMANCE

Table 1 below provides the performance of the resonator (labeled in the source code as "Resonator") in terms of both timing and memory requirements. Since a double precision multiplier is used (software implementation), the multiplier timing is not always constant. Therefore the timings are given for the worst case. Note that irrespective of the frequency and the sampling (resolution) of the tone, program memory requirements is only 54 locations which in case of table lookup could be very large.

TABLE 1

Cycles	235 Cycles
Time @ 16 MHz	58.75 µs
Time @ 25 MHz	47 µs
Data Memory	9 + 9*(# of tones to be generated)
Program memory	54 locations

APPLICATIONS

Tone generation is required in many applications. The code provided in this application note may be used as a general purpose routine to generate desired tones. It can be used in applications involving secure off-site control, where commands/data in the format of tones are transmitted over a telephone line. The tone generation finds applications involving signal modulations as well. The routine can be used to generate audible tones and output to a speaker connected to an I/O Port or a PWM channel.

Author: Amar Palacherla
Logic Products Division

© 1993 Microchip Technology Inc.

APPENDIX A: TONE.LST

```
MPASM B0.54                                                    PAGE  1


;*****************************************************************************
;                       Dual Tone Generation
;
;       A generic resonator subroutine is implemented to generate
; tones. Samples Of A Sin/Cos Wave are generated using
; recursive techniques. Table Lookups are thus avoided
; This is especially useful in generating multiple tones
; (e.g. DTMF tone generation, tone signalling, etc.), or programmable
; tone generation which may vary for each application unit.
;
;*****************************************************************************
;
        LIST    P=17C42, C=80, T=ON, R=DEC, N=0
        include "17c42.h"


        include "17c42.mac"


;
;*************************************************************

;       TBLADDR
;
; DESCRIPTION:
;       Load 16 bit table pointer with specified label
;
; TIMING (in cycles):
;       4
;

TBLADDR MACRO   label

        MOVLW   (label)
        MOVWF   tblptrl
        MOVLW   page    (label)
        MOVWF   tblptrh

        ENDM

;*************************************************************

;       ADDLBL
;
; DESCRIPTION:
;       Add A Label (16 bit constant) To A File Register (1

;
; TIMING (in cycles):
;       4
;

ADDLBL  MACRO   label,f

        MOVLW   (label)
        ADDWF   f+B0
        MOVLW   page    (label)
        ADDWFC  f+B1

        ENDM
```

© 1993 Microchip Technology Inc.

```
                        ;*******************************************************

                        ;
                                CBLOCK  0
0000 0004                       B0,B1,B2,B3         ; RAM offset constants
                                ENDC
                        ;
                                CBLOCK  0x18
0018 0002                       AARG,AARG1          ; 16 bit multiplier A
001A 0002                       BARG,BARG1          ; 16 bit multiplicand B
001C 0004                       DPX,DPX1,DPX2,DPX3; 32 bit multiplier result = A*B

                                ENDC
                                                    ;
                                CBLOCK              ; Generic Resonator

0020 0002                       K1,K11              ; K1
0022 0003                       SinCos_2C,SinCos_2,SinCos1_2
                                                    ; y(n-2), Init Value = K3

0025 0002                       SinCos_1,SinCos1_1; y(n-1), Init Value = K2

0027 0002                       SinCos,SinCos1      ; y(n)
0029 0000
                                ENDC

                                CBLOCK
0029 0002                       tone1K1,tone1K11    ; Tone 1 variables
002B 0003                       tone1_2C,tone1_2,tone11_2
002E 0002                       tone1_1,tone11_1
0030 0002                       tone1,tone11
0032 0000
0032 0002                       tone2K1,tone2K11    ; Tone 2 variables
0034 0003                       tone2_2C,tone2_2,tone21_2
0037 0002                       tone2_1,tone21_1
0039 0002                       tone2,tone21
003B 0000
003B 0002                       dualTone,dualTone1; tone1+tone2
       ENDC


                        ;*******************************************************


005A                            #define ADD_OFFSET  FALSE

005B                            #define coeff_addr  0x0030 ; prog mem addr of

005C                            #define DummyAddr   0x0A00

                        ;*******************************************************
                        ;               Test Routine For Sin Wave Generation
                        ;
                        ;       (a) Call Initialization Subroutine to read desired s

                        ;               freq from prog mem to data mem.
                        ;       (b) Call the resonator subroutine to generate sampl

                        ;               sine wave.
                        ;       (c) Perform a Dummy Table Write of The Sine Wave Da

                        ;               PICMASTER emulator can capture the data in

                        ;
                        ;*******************************************************
```


© 1993 Microchip Technology Inc.

```
                        ORG     0x0000
0000 C040               goto    start

                ;*******************************************************

                        ;Resonator Co-efficients For Desired Tones

                        ORG     coeff_addr

                        ; Tone 1 Resonator Constants
                        ;Sample Rate = fs = 8 kHz, Tone Freq = f = 0.800 Khz

0030 678E               DATA    26510   ; K1 = COS(wT) = COS(360*f/

0031 259E               DATA    9630    ; K2 = SIN(wT) = SIN(360*f/

0032 0000               DATA    0       ; K3 is init value of y(n-2


                        ; Tone 2 Resonator Constants
                        ; Sample Rate = fs = 8 khz, Tone Freq = f = 1.10 Khz

0033 5321               DATA    21281   ; K1 = COS(wT) = COS(360*f/

0034 1855               DATA    6229    ; K2 = SIN(wT) = SIN(360*f/

0035 0000               DATA    0       ; K3 is init value of y(n-2


                ;*******************************************************


                        ORG     0x0040
                start
0040 E057               call    Coeff_Read
                                        ; load table pointers with a dummy  addr

                        MOVK16  DummyAddr,tblptrl

0041 B000               MOVLW   (0x0A00) & 0xff
0042 010D               MOVWF   tblptrl+B0
0043 B00A               MOVLW   ((0x0A00) >> 8)
0044 010E               MOVWF   tblptrl+B1


                NextSample
0045 0000               nop
0046 A43B               tlwt    0,dualTone
                capture
0047 AE3C               tablwt  1,0,dualTone+B1     ; for PICMASTER tr

0048 0000               nop

0049 B029               movlw   tone1K1             ; load indirect add

004A 0101               movwf   fsr0                ; for Tone 1
004B E06C               call    Resonator           ; Compute next samp


004C B032               movlw   tone2K1             ; load indirect add

004D 0101               movwf   fsr0                ; for Tone 2
004E E06C               call    Resonator           ; compute next samp


                                        ; Compute Tone1 + Tone2 for dual tone

                        ADD16ACC        tone1,tone2,dualTone
```

© 1993 Microchip Technology Inc.

```
                                                    ; dualTone = tone1
004F 6030                   movfp   tone1+B0,wreg
0050 0E39                   addwf   tone2+B0,w
0051 013B                   movwf   dualTone+B0
0052 6031                   movfp   tone1+B1,wreg
0053 103A                   addwfc  tone2+B1,w
0054 013C                   movwf   dualTone+B1
0055 C045                   goto    NextSample

0056 C056             self  goto    self

                    ;*******************************************************

                    ; Initialization routine :
                    ; Read Tone 1 & Tone 2 Resonator Frequencies from Program M

                    ; Data Memory
                    ;
                    ;       Program Memory  :       3 + 8*(#of tones to be gene

                    ;       Timing          :       4 + 11*(#of tones to be gen
                    ;*******************************************************
                    Coeff_Read

                            TBLADDR coeff_addr
0057 B030                   MOVLW   (0x0030)
0058 010D                   MOVWF   tblptrl
0059 B000                   MOVLW   page    (0x0030)
005A 010E                   MOVWF   tblptrh
005B A929                   tablrd  0,1,tone1K1
005C A029                   tlrd    0,tone1K1
005D AB2A                   tablrd  1,1,tone1K1+B1  ; read K1
005E A02E                   tlrd    0,tone1_1
005F AB2F                   tablrd  1,1,tone1_1+B1  ; read K2
0060 A02C                   tlrd    0,tone1_2
0061 A22D                   tlrd    1,tone1_2+B1    ; read K3
0062 292B                   clrf    tone1_2C
0063 A932                   tablrd  0,1,tone2K1
0064 A032                   tlrd    0,tone2K1
0065 AB33                   tablrd  1,1,tone2K1+B1  ; read K1
0066 A037                   tlrd    0,tone2_1
0067 AB38                   tablrd  1,1,tone2_1+B1  ; read K2
0068 A035                   tlrd    0,tone2_2
0069 A236                   tlrd    1,tone2_2+B1    ; read K3
006A 2934                   clrf    tone2_2C
006B 0002                   return

                    ;*******************************************************
                    ;                       Resonator Subroutine
                    ;
                    ; Before calling this routine, load the indirect register,
                    ; with the starting RAM address of the desired Tone Variabl
                    ; ( eg. For Tone1 Generation, load FSR0 with "Tone1K1" addr
                    ;
                    ;
                    ; Timing (worst case) :
                    ;               20 + 36 + (worst case multiplier time)
                    ;               = 56 + 179 = 235 cycles
                    ;               = 58.75 uS @ 16Mhz
                    ;               = 47.00 uS @ 20Mhz
                    ;               = 37.60 uS @ 25 Mhz
                    ;
                    ; Memory Requirements :
                    ;               Program Memory  : 54 locations
                    ;               Data Memory     : 9 + 9*(#of tones to be ge
                    ;
                    ;*******************************************************
```


© 1993 Microchip Technology Inc.

```
                        ;
                        Resonator
                        ;
                        ; Transfer tone varialbles to resonator's variables using i
                        ; This is necessary for making the "Resonator" a generic
                        ; subroutine, so that the same code can be called for vario
                        ; tones.
                        ;  Indirect addressing mode can be used throught the code i
                        ; subroutine, but is less efficient.
                        ;
006C 8D04                       bcf     _fs1
006D 8404                       bsf     _fs0            ; auto increment FSR0
006E 4020                       movpf   indf0,K1+B0
006F 4021                       movpf   indf0,K1+B1
0070 4022                       movpf   indf0,SinCos_2C
0071 4023                       movpf   indf0,SinCos_2+B0
0072 4024                       movpf   indf0,SinCos_2+B1
0073 4025                       movpf   indf0,SinCos_1+B0
0074 4026                       movpf   indf0,SinCos_1+B1
0075 4027                       movpf   indf0,SinCos+B0
0076 4028                       movpf   indf0,SinCos+B1
                                ;
                        ; Compute 2*K1*y(n-1)
                        ;
                                MOVFP16 K1,BARG
0077 7A20                       MOVFP   K1+B0,BARG+B0       ; move K1(B0) t
0078 7B21                       MOVFP   K1+B1,BARG+B1       ; move K1(B1) t
                                MOVFP16 SinCos_1,AARG
0079 7825                       MOVFP   SinCos_1+B0,AARG+B0; move Si
007A 7926                       MOVFP   SinCos_1+B1,AARG+B1; move Si
007B E0A2                       call    DblMult
007C 8804                       BCF     _carry
007D 1B1C                       RLCF    DPX+B0
007E 1B1D                       RLCF    DPX+B1
007F 1B1E                       RLCF    DPX+B2
0080 1B1F                       RLCF    DPX+B3              ;
                        ; subtract y(n-2)*(2**15)           ;
0081 2922                       clrf    SinCos_2C
                                RRC24   SinCos_2C
0082 1A24                       RLCF    SinCos_2C+B2,W      ; move sign
0083 1924                       RRCF    SinCos_2C+B2
0084 1923                       RRCF    SinCos_2C+B1
0085 1922                       RRCF    SinCos_2C+B0
                                SUB24   SinCos_2C,DPX1      ; DPX = 2*K1*y(n-1) - y(n-
0086 6022                       MOVFP   SinCos_2C+B0,wreg   ; get lowes
0087 051D                       SUBWF   DPX1+B0             ; sub lowest byt
0088 6023                       MOVFP   SinCos_2C+B1,wreg   ; get 2nd b
0089 031E                       SUBWFB  DPX1+B1             ; sub 2nd byte o
008A 6024                       MOVFP   SinCos_2C+B2,wreg   ; get 3rd b
008B 031F                       SUBWFB  DPX1+B2             ; sub 3rd byte o
                                RLC24   DPX1                ; adjust decimal point
008C 8804                       BCF     _carry
008D 1B1D                       RLCF    DPX1+B0
008E 1B1E                       RLCF    DPX1+B1
008F 1B1F                       RLCF    DPX1+B2
                                ;
                        ; update past samples with newly computed values
                        ;
                                MOVPF16 DPX2,SinCos         ; y(n) = 2*K1*y(n-1) - y(n
0090 5E27                       MOVPF   DPX2+B0,SinCos+B0   ; move DPX2
0091 5F28                       MOVPF   DPX2+B1,SinCos+B1   ; move DPX2
                                MOV16   SinCos_1,SinCos_2   ; y(n-2) = y(n-1)
0092 6025                       MOVFP   SinCos_1+B0,wreg    ; get byte o
0093 0123                       MOVWF   SinCos_2+B0         ; move to Si
0094 6026                       MOVFP   SinCos_1+B1,wreg    ; get byte o
0095 0124                       MOVWF   SinCos_2+B1         ; move to Si
                                MOVPF16 DPX2,SinCos_1       ; y(n-1) = y(n)
0096 5E25                       MOVPF   DPX2+B0,SinCos_1+B0; move DP
0097 5F26                       MOVPF   DPX2+B1,SinCos_1+B1; move DP
```

© 1993 Microchip Technology Inc.

```
                        ;
                        ; Generation Of The Next Sample Of The Resonator (sine wave
                        ; The 16 bit result is stored in location "SinCos" (low Byt
                        ; "SinCos+1" (High Byte)
                        ;
                        ; write back all the computed values to respective tone var
                        ;
0098 0701                       decf    fsr0
0099 8D04                       bcf     _fs1
009A 8C04                       bcf     _fs0
009B 6028                       movfp   SinCos+B1,indf0
009C 6027                       movfp   SinCos+B0,indf0
009D 6026                       movfp   SinCos_1+B1,indf0
009E 6025                       movfp   SinCos_1+B0,indf0
009F 6024                       movfp   SinCos_2C+B2,indf0
00A0 6023                       movfp   SinCos_2C+B1,indf0                      ;
00A1 0002                       return
                        ;
                        ;*********************************************************

                        ;       Include Double Precision Multiplication Routine
                        ;*********************************************************
0001                    SIGNED  equ     TRUE
                                include "17c42mpy.mac"
                        ;       NOLIST
                        ;*********************************************************
                        ;
                        ;*********************************************************
                        ;               Double Precision Multiplier For PIC17C42
                        ;
                        ;       Dmult
                        ;
                        ; DESCRIPTION:
                        ;
                        ;   Multiplication : AARG (16 bits) * BARG (16 bits) -> DPX
                        ;
                        ;       (a) Load the 1st operand in locations AARG+B0 & AARG
                        ;       (b) Load the 2nd operand in locations BARG+B0 & BARG
                        ;       (c) CALL Dmult
                        ;       (d) The 32 bit result is in locations ( DPX+B0,DPX+B
                        ;
                        ;       In the signed case, a savings of 9 clks can be real
                        ;       BARG as the positive factor in the product when pos
                        ; TIMING (worst case):
                        ;                       unsigned:                       17
                        ;
                        ;                       signed:                 BARG+   17
                        ;                                               BARG-   17
                        ;
                        ;   NOTE :   Define SIGNED/UNSIGNED To 1/0 before including
                        ;               this file in your program
                        ;
                        ;*********************************************************
                        ;               Multiplication Macro
                        ;*********************************************************
                        ;
                        ; TIMING:       unsigned:     11+7*10+8*11   = 169 clks
                        ;(worst case)   signed:       11+7*10+7*11+5 = 163 clks
                        ;
                        MULTMAC  MACRO
                                variable i
                                i = 0
                                if   SIGNED
                                while i < 15
                                else
                                while i < 16
                                endif
                                if i < 8                ; test low byte
```


© 1993 Microchip Technology Inc.

```
      btfsc     BARG+B0,i
        else                ; test high byte
      btfsc     BARG+B1,i-8
    fi
      goto    add#v(i)
    if i < 8
      rlcf    DPX+B3,W      ; rotate sign into carry bit
      rrcf    DPX+B3        ; for i < 8, no meaningful bits
      rrcf    DPX+B2        ; are in DPX+B0
      rrcf    DPX+B1
    else
      rlcf    DPX+B3,W      ; rotate sign into carry bit
      rrcf    DPX+B3
      rrcf    DPX+B2
      rrcf    DPX+B1
      rrcf    DPX+B0
    fi
      i = i+1
    endw
      clrf    DPX+B0        ; if we get here, BARG = 0
      return
          add0
      movfp   AARG+B0,WREG
      addwf   DPX+B2        ;add lsb
      movfp   AARG+B1,WREG
      addwfc  DPX+B3        ;add msb
      rlcf    AARG+B1,W     ; rotate sign into carry bit
      rrcf    DPX+B3        ; for i < 8, no meaningful bits
      rrcf    DPX+B2        ; are in DPX+B0
      rrcf    DPX+B1
i = 1
if   SIGNED
       while i < 15
else
   while i < 16
endif
   if i < 8
     btfss    BARG+B0,i    ;test low byte
   else
     btfss    BARG+B1,i-8  ; test high byte
   fi
     goto    noadd#v(i)
         add#v(i)
     movfp   AARG+B0,WREG
     addwf   DPX+B2        ;add lsb
     movfp   AARG+B1,WREG
     addwfc  DPX+B3        ;add msb
          noadd#v(i)
   if i < 8
     rlcf    AARG+B1,W     ; rotate sign into carry bit
     rrcf    DPX+B3        ; for i < 8, no meaningful bits
     rrcf    DPX+B2        ; are in DPX+B0
     rrcf    DPX+B1
   else
     rlcf    AARG+B1,W     ; rotate sign into carry bit
     rrcf    DPX+B3
     rrcf    DPX+B2
     rrcf    DPX+B1
     rrcf    DPX+B0
       fi
     i = i+1
       endw
if   SIGNED
     rlcf    AARG+B1,W     ; since BARG is always made posit
     rrcf    DPX+B3        ; the last bit is known to be zer
     rrcf    DPX+B2
     rrcf    DPX+B1
     rrcf    DPX+B0
```


© 1993 Microchip Technology Inc.

```
                        endif

                           ENDM

                    ;              Double Precision Multiply ( 16x16 -> 32 )
                    ;          ( AARG*BARG -> : 32 bit output in DPX
                    ;
                           DblMult
                        if   SIGNED

00A2 971B                  btfss   BARG+B1,MSB        ; test sign of BARG

00A3 C0AE                  goto    argsok             ; if positive, ok
                    NEG16  AARG+B0                    ; if negative, then negate


00A4 1318                  COMF    AARG+B0+B0
00A5 1319                  COMF    AARG+B0+B1
00A6 2900                  CLRF    wreg
00A7 1518                  INCF    AARG+B0+B0
00A8 1119                  ADDWFC  AARG+B0+B1


                    NEG16   BARG+B0                   ; AARG and BARG

00A9 131A                  COMF    BARG+B0+B0
00AA 131B                  COMF    BARG+B0+B1
00AB 2900                  CLRF    wreg
00AC 151A                  INCF    BARG+B0+B0
00AD 111B                  ADDWFC  BARG+B0+B1



                        endif

                    argsok
                           CLR16   DPX+B2             ; clear initial partial pr


00AE 291E                  CLRF    DPX+B2+B0
00AF 291F                  CLRF    DPX+B2+B1

                           MULTMAC                    ; use macro for multiplica

0000                       variable i

0000                       i = 0

                           if   SIGNED

                           while i < 15

                        else

                           while i < 16

                        endif
                           if i < 8                   ; test low byte

                           btfsc      BARG+B0,i

                        else                          ; test high byte
                           btfsc      BARG+B1,i-8
                        fi
                           goto    add#v(i)
                           if i < 8
                           rlcf    DPX+B3,W           ; rotate sign into carry bit
                           rrcf    DPX+B3             ; for i < 8, no meaningful bits
```


© 1993 Microchip Technology Inc.

```
                                    rrcf    DPX+B2      ; are in DPX+B0
                                    rrcf    DPX+B1
                                else
                                    rlcf    DPX+B3,W    ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                fi
                                    i = i+1
                                 endw
                                if i < 8                ; test low byte
00B0 981A                           btfsc   BARG+B0,i
                                    else                ; test high byte
                                    btfsc   BARG+B1,i-8
                                fi
00B1 C113                           goto    add0
                                if i < 8
00B2 1A1F                           rlcf    DPX+B3,W    ; rotate sign into
00B3 191F                           rrcf    DPX+B3      ; for i < 8, no mea
00B4 191E                           rrcf    DPX+B2      ; are in DPX+B0
00B5 191D                           rrcf    DPX+B1
                                else
                                    rlcf    DPX+B3,W    ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                fi
0001                                i = i+1
                                if i < 8                ; test low byte
00B6 991A                           btfsc   BARG+B0,i
                                else                    ; test high byte
                                    btfsc   BARG+B1,i-8
                                fi
00B7 C11D                           goto    add1
                                if i < 8
00B8 1A1F                           rlcf    DPX+B3,W    ; rotate sign into
00B9 191F                           rrcf    DPX+B3      ; for i < 8, no mea
00BA 191E                           rrcf    DPX+B2      ; are in DPX+B0
00BB 191D                           rrcf    DPX+B1
                                else
                                    rlcf    DPX+B3,W    ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                fi
0002                                i = i+1
                                if i < 8                ; test low byte
00BC 9A1A                           btfsc     BARG+B0,i
                                else                    ; test high byte
                                    btfsc     BARG+B1,i-8
                                fi
00BD C127                           goto    add2
                                if i < 8
00BE 1A1F                           rlcf    DPX+B3,W    ; rotate sign into
00BF 191F                           rrcf    DPX+B3      ; for i < 8, no mea
00C0 191E                           rrcf    DPX+B2      ; are in DPX+B0
00C1 191D                           rrcf    DPX+B1
                                else
                                    rlcf    DPX+B3,W    ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                fi
0003                                i = i+1
                                if i < 8                ; test low byte
```

© 1993 Microchip Technology Inc.

```
                                   else                ; test high byte
                                   btfsc      BARG+B1,i-8
                                fi
00C3 C131                          goto       add3
                                if i < 8
00C4 1A1F                          rlcf       DPX+B3,W    ; rotate sign into
00C5 191F                          rrcf       DPX+B3      ; for i < 8, no mea
00C6 191E                          rrcf       DPX+B2      ; are in DPX+B0
00C7 191D                          rrcf       DPX+B1
                                   else
                                   rlcf       DPX+B3,W    ; rotate sign into carry bit
                                   rrcf       DPX+B3
                                   rrcf       DPX+B2
                                   rrcf       DPX+B1
                                   rrcf       DPX+B0
                                fi
0004                               i = i+1
                                if i < 8                   ; test low byte
00C8 9C1A                          btfsc      BARG+B0,i
                                   else                    ; test high byte
                                   btfsc      BARG+B1,i-8
                                fi
00C9 C13B                          goto       add4
                                if i < 8
00CA 1A1F                          rlcf       DPX+B3,W    ; rotate sign into
00CB 191F                          rrcf       DPX+B3      ; for i < 8, no mea
00CC 191E                          rrcf       DPX+B2      ; are in DPX+B0
00CD 191D                          rrcf       DPX+B1
                                else
                                   rlcf       DPX+B3,W    ; rotate sign into carry bit
                                   rrcf       DPX+B3
                                   rrcf       DPX+B2
                                   rrcf       DPX+B1
                                   rrcf       DPX+B0
                                fi
0005                               i = i+1
                                if i < 8                   ; test low byte
00CE 9D1A                          btfsc      BARG+B0,i
                                    else                   ; test high byte
                                   btfsc      BARG+B1,i-8
                                fi
00CF C145                          goto       add5
                                if i < 8
00D0 1A1F                          rlcf       DPX+B3,W    ; rotate sign into
00D1 191F                          rrcf       DPX+B3      ; for i < 8, no mea
00D2 191E                          rrcf       DPX+B2      ; are in DPX+B0
00D3 191D                          rrcf       DPX+B1
                                else
                                   rlcf       DPX+B3,W    ; rotate sign into carry bit
                                   rrcf       DPX+B3
                                   rrcf       DPX+B2
                                   rrcf       DPX+B1
                                   rrcf       DPX+B0
                                fi
006                                i = i+1
                                if i < 8                   ; test low byte
00D4 9E1A                          btfsc      BARG+B0,i
                                    else                    ; test high byte
                                   btfsc      BARG+B1,i-8
                                fi
0D5 C14F                           goto       add6
                                if i < 8
00D6 1A1F                          rlcf       DPX+B3,W    ; rotate sign into
00D7 191F                          rrcf       DPX+B3      ; for i < 8, no mea
00D8 191E                          rrcf       DPX+B2      ; are in DPX+B0
00D9 191D                          rrcf       DPX+B1
                                else
                                   rlcf       DPX+B3,W    ; rotate sign into carry bit
```


© 1993 Microchip Technology Inc.

```
                                  rrcf    DPX+B3
                                  rrcf    DPX+B2
                                  rrcf    DPX+B1
                                  rrcf    DPX+B0
                               fi
0007                              i = i+1
                               if i < 8                ; test low byte
00DA 9F1A                         btfsc   BARG+B0,i
                                    else               ; test high byte
                                  btfsc   BARG+B1,i-8
                               fi
00DB C159                         goto    add7
                               if i < 8
00DC 1A1F                         rlcf    DPX+B3,W    ; rotate sign into
00DD 191F                         rrcf    DPX+B3      ; for i < 8, no mea
00DE 191E                         rrcf    DPX+B2      ; are in DPX+B0
00DF 191D                         rrcf    DPX+B1
                               else
                                  rlcf    DPX+B3,W    ; rotate sign into carry bit
                                  rrcf    DPX+B3
                                  rrcf    DPX+B2
                                  rrcf    DPX+B1
                                  rrcf    DPX+B0
                               fi
0008                              i = i+1
                               if i < 8                ; test low byte
                                  btfsc   BARG+B0,i
                                    else               ; test high byte
00E0 981B                         btfsc   BARG+B1,i-8
                               fi
00E1 C163                         goto    add8
                               if i < 8
                                  rlcf    DPX+B3,W    ; rotate sign into carry bit
                                  rrcf    DPX+B3      ; for i < 8, no meaningful bits
                                  rrcf    DPX+B2      ; are in DPX+B0
                                  rrcf    DPX+B1
                               else
00E2 1A1F                         rlcf    DPX+B3,W    ; rotate sign into
00E3 191F                         rrcf    DPX+B3
00E4 191E                         rrcf    DPX+B2
00E5 191D                         rrcf    DPX+B1
00E6 191C                         rrcf    DPX+B0
                               fi
0009                              i = i+1
                               if i < 8                ; test low byte
                                  btfsc   BARG+B0,i
                                    else               ; test high byte
00E7 991B                         btfsc   BARG+B1,i-8
                               fi
00E8 C16E                         goto    add9
                               if i < 8
                                  rlcf    DPX+B3,W    ; rotate sign into carry bit
                                  rrcf    DPX+B3      ; for i < 8, no meaningful bits
                                  rrcf    DPX+B2      ; are in DPX+B0
                                  rrcf    DPX+B1
                               else
00E9 1A1F                         rlcf    DPX+B3,W    ; rotate sign into
00EA 191F                         rrcf    DPX+B3
00EB 191E                         rrcf    DPX+B2
00EC 191D                         rrcf    DPX+B1
00ED 191C                         rrcf    DPX+B0
                               fi
000A                              i = i+1
                               if i < 8                ; test low byte
                                  btfsc   BARG+B0,i
                                    else               ; test high byte
00EE 9A1B                         btfsc   BARG+B1,i-8
                               fi
00EF C179                         goto    add10
```


© 1993 Microchip Technology Inc.

```
                                 if i < 8
                                    rlcf    DPX+B3,W   ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
00F0 1A1F                           rlcf    DPX+B3,W   ; rotate sign into
00F1 191F                           rrcf    DPX+B3
00F2 191E                           rrcf    DPX+B2
00F3 191D                           rrcf    DPX+B1
00F4 191C                           rrcf    DPX+B0
                                 fi
000B                                i = i+1
                                 if i < 8                ; test low byte
                                    btfsc      BARG+B0,i
                                     else                ; test high byte
00F5 9B1B                           btfsc      BARG+B1,i-8
                                 fi
00F6 C184                           goto    add11
                                 if i < 8
                                    rlcf    DPX+B3,W   ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
00F7 1A1F                           rlcf    DPX+B3,W   ; rotate sign into
00F8 191F                           rrcf    DPX+B3
00F9 191E                           rrcf    DPX+B2
00FA 191D                           rrcf    DPX+B1
00FB 191C                           rrcf    DPX+B0
                                 fi
000C                                i = i+1
                                 if i < 8                ; test low byte
                                    btfsc      BARG+B0,i
                                     else                ; test high byte
00FC 9C1B                           btfsc      BARG+B1,i-8
                                 fi
00FD C18F                           goto    add12

                                 if i < 8
                                    rlcf    DPX+B3,W   ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
00FE 1A1F                           rlcf    DPX+B3,W   ; rotate sign into
00FF 191F                           rrcf    DPX+B3
0100 191E                           rrcf    DPX+B2
0101 191D                           rrcf    DPX+B1
0102 191C                           rrcf    DPX+B0
                                 fi
000D                                i = i+1
                                 if i < 8                ; test low byte
                                    btfsc      BARG+B0,i
                                      else               ; test high byte
0103 9D1B                           btfsc      BARG+B1,i-8
                                 fi
0104 C19A                           goto    add13
                                 if i < 8
                                    rlcf    DPX+B3,W   ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
0105 1A1F                           rlcf    DPX+B3,W   ; rotate sign into
0106 191F                           rrcf    DPX+B3
0107 191E                           rrcf    DPX+B2
0108 191D                           rrcf    DPX+B1
0109 191C                           rrcf    DPX+B0
```


© 1993 Microchip Technology Inc.

```
                                fi
000E                              i = i+1
                                if i < 8                    ; test low byte
                                   btfsc   BARG+B0,i
                                   else                   ; test high byte
010A 9E1B                          btfsc   BARG+B1,i-8
                                fi
010B C1A5                          goto    add14
                                if i < 8
                                   rlcf    DPX+B3,W         ; rotate sign into carry bit
                                   rrcf    DPX+B3           ; for i < 8, no meaningful bits
                                   rrcf    DPX+B2           ; are in DPX+B0
                                   rrcf    DPX+B1
                                else
010C 1A1F                          rlcf    DPX+B3,W         ; rotate sign into
010D 191F                          rrcf    DPX+B3
010E 191E                          rrcf    DPX+B2
010F 191D                          rrcf    DPX+B1
0110 191C                          rrcf    DPX+B0
                                fi
000F                               i = i+1
0111 291C                          clrf    DPX+B0           ; if we get here, B
0112 0002                          return
                                add0
0113 6018                          movfp   AARG+B0,WREG
0114 0F1E                          addwf   DPX+B2           ;add lsb
0115 6019                          movfp   AARG+B1,WREG
0116 111F                          addwfc  DPX+B3           ;add msb
0117 1A19                          rlcf    AARG+B1,W        ; rotate sign into
0118 191F                          rrcf    DPX+B3           ; for i < 8, no mea
0119 191E                          rrcf    DPX+B2           ; are in DPX+B0
011A 191D                          rrcf    DPX+B1
0001                            i = 1
                                if   SIGNED
                                     while i < 15
                                else
                                     while i < 16
                                endif
                                if i < 8
                                    btfss     BARG+B0,i     ;test low byte
                                else
                                    btfss     BARG+B1,i-8   ; test high byte
                                fi
                                    goto    noadd#v(i)
                                add#v(i)
                                    movfp   AARG+B0,WREG
                                    addwf   DPX+B2           ;add lsb
                                    movfp   AARG+B1,WREG
                                    addwfc  DPX+B3           ;add msb
                                    noadd#v(i)
                                 if i < 8
                                    rlcf    AARG+B1,W        ; rotate sign into carry bit
                                    rrcf    DPX+B3           ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2           ; are in DPX+B0
                                    rrcf    DPX+B1
                                else
                                    rlcf    AARG+B1,W        ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                 fi
                                i = i+1
                                endw
                                if i < 8                     ;test low byte
011B 911A                          btfss     BARG+B0,i
                                else
                                                           ; test high byte
                                    btfss     BARG+B1,i-8
```

```
                                    fi
011C C121                           goto    noadd1
                    add1
011D 6018                           movfp   AARG+B0,WREG
011E 0F1E                           addwf   DPX+B2      ;add lsb
011F 6019                           movfp   AARG+B1,WREG
0120 111F                           addwfc  DPX+B3      ;add msb
                    noadd1
                                    if i < 8
0121 1A19                           rlcf    AARG+B1,W  ; rotate sign into
0122 191F                           rrcf    DPX+B3     ; for i < 8, no mea
0123 191E                           rrcf    DPX+B2     ; are in DPX+B0
0124 191D                           rrcf    DPX+B1
                                    else
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                      fi
0002                                i = i+1
                                    if i < 8           ;test low byte
0125 921A                           btfss       BARG+B0,i
                                    else                    ; test high byte
                                    btfss       BARG+B1,i-8
                                    fi
0126 C12B                           goto    noadd2
                    add2
0127 6018                           movfp   AARG+B0,WREG
0128 0F1E                           addwf   DPX+B2      ;add lsb
0129 6019                           movfp   AARG+B1,WREG
012A 111F                           addwfc  DPX+B3      ;add msb
                    noadd2
                                    if i < 8
012B 1A19                           rlcf    AARG+B1,W  ; rotate sign into
012C 191F                           rrcf    DPX+B3     ; for i < 8, no mea
012D 191E                           rrcf    DPX+B2     ; are in DPX+B0
012E 191D                           rrcf    DPX+B1
                                    else
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                      fi
0003                                i = i+1
                                    if i < 8                ;test low byte
012F 931A                           btfss       BARG+B0,i
                                    else
                                                            ; test high byte
                                    btfss       BARG+B1,i-8
                                    fi
0130 C135                           goto    noadd3
                    add3
0131 6018                           movfp   AARG+B0,WREG
0132 0F1E                           addwf   DPX+B2      ;add lsb
0133 6019                           movfp   AARG+B1,WREG
0134 111F                           addwfc  DPX+B3      ;add msb
                    noadd3
                                    if i < 8
0135 1A19                           rlcf    AARG+B1,W  ; rotate sign into
0136 191F                           rrcf    DPX+B3     ; for i < 8, no mea
0137 191E                           rrcf    DPX+B2     ; are in DPX+B0
0138 191D                           rrcf    DPX+B1
                                    else
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
```


© 1993 Microchip Technology Inc.

```
                                    rrcf    DPX+B0
                                      fi
0004                                  i = i+1
                                    if i < 8          ;test low byte
0139 941A                             btfss   BARG+B0,i
                                    else
                                                      ; test high byte
                                      btfss   BARG+B1,i-8
                                    fi
013A C13F                             goto    noadd4
                            add4
013B 6018                             movfp   AARG+B0,WREG
013C 0F1E                             addwf   DPX+B2    ;add lsb
013D 6019                             movfp   AARG+B1,WREG
013E 111F                             addwfc  DPX+B3    ;add msb
                            noadd4
                                    if i < 8
013F 1A19                             rlcf    AARG+B1,W  ; rotate sign into
0140 191F                             rrcf    DPX+B3     ; for i < 8, no mea
0141 191E                             rrcf    DPX+B2     ; are in DPX+B0
0142 191D                             rrcf    DPX+B1
                                    else
                                      rlcf    AARG+B1,W  ; rotate sign into carry bit
                                      rrcf    DPX+B3
                                      rrcf    DPX+B2
                                      rrcf    DPX+B1
                                      rrcf    DPX+B0
                                        fi
0005                                  i = i+1
                                    if i < 8          ;test low byte
0143 951A                             btfss   BARG+B0,i
                                    else              ; test high byte
                                      btfss   BARG+B1,i-8
                                    fi
0144 C149                             goto    noadd5
                            add5
0145 6018                             movfp   AARG+B0,WREG
0146 0F1E                             addwf   DPX+B2    ;add lsb
0147 6019                             movfp   AARG+B1,WREG
0148 111F                             addwfc  DPX+B3    ;add msb
                            noadd5
                                    if i < 8
0149 1A19                             rlcf    AARG+B1,W  ; rotate sign into
014A 191F                             rrcf    DPX+B3     ; for i < 8, no mea
014B 191E                             rrcf    DPX+B2     ; are in DPX+B0
014C 191D                             rrcf    DPX+B1
                                    else
                                      rlcf    AARG+B1,W  ; rotate sign into carry bit
                                      rrcf    DPX+B3
                                      rrcf    DPX+B2
                                      rrcf    DPX+B1
                                      rrcf    DPX+B0
                                        fi
0006                                  i = i+1
                                    if i < 8          ;test low byte
014D 961A                             btfss   BARG+B0,i
                                    else              ; test high byte
                                      btfss   BARG+B1,i-8
                                    fi
014E C153                             goto    noadd6
                            add6
014F 6018                             movfp   AARG+B0,WREG
0150 0F1E                             addwf   DPX+B2    ;add lsb
0151 6019                             movfp   AARG+B1,WREG
0152 111F                             addwfc  DPX+B3    ;add msb
                            noadd6
                                    if i < 8
0153 1A19                             rlcf    AARG+B1,W  ; rotate sign into
0154 191F                             rrcf    DPX+B3     ; for i < 8, no mea
```

© 1993 Microchip Technology Inc.

```
0155 191E                          rrcf    DPX+B2      ; are in DPX+B0
0156 191D                          rrcf    DPX+B1
                                 else
                                   rlcf    AARG+B1,W   ; rotate sign into carry bit
                                   rrcf    DPX+B3
                                   rrcf    DPX+B2
                                   rrcf    DPX+B1
                                   rrcf    DPX+B0
                                     fi
0007                               i = i+1
                                 if i < 8                ;test low byte

0157 971A                          btfss   BARG+B0,i
                                 else                    ; test high byte
                                   btfss   BARG+B1,i-8
                                 fi
0158 C15D                          goto    noadd7
                     add7
0159 6018                          movfp   AARG+B0,WREG
015A 0F1E                          addwf   DPX+B2      ;add lsb
015B 6019                          movfp   AARG+B1,WREG
015C 111F                          addwfc  DPX+B3      ;add msb
                     noadd7
                                 if i < 8
015D 1A19                          rlcf    AARG+B1,W   ; rotate sign into
015E 191F                          rrcf    DPX+B3      ; for i < 8, no mea
015F 191E                          rrcf    DPX+B2      ; are in DPX+B0
0160 191D                          rrcf    DPX+B1
                                 else
                                   rlcf    AARG+B1,W   ; rotate sign into carry bit
                                   rrcf    DPX+B3
                                   rrcf    DPX+B2
                                   rrcf    DPX+B1
                                   rrcf    DPX+B0
                                     fi
0008                               i = i+1
                                 if i < 8                ;test low byte
                                   btfss   BARG+B0,i
                                 else                    ; test high byte
0161 901B                          btfss   BARG+B1,i-8
                                 fi
0162 C167                          goto    noadd8
                     add8
0163 6018                          movfp   AARG+B0,WREG
0164 0F1E                          addwf   DPX+B2      ;add lsb
0165 6019                          movfp   AARG+B1,WREG
0166 111F                          addwfc  DPX+B3      ;add msb
                     noadd8
                                 if i < 8
                                   rlcf    AARG+B1,W   ; rotate sign into carry bit
                                   rrcf    DPX+B3      ; for i < 8, no meaningful bits
                                   rrcf    DPX+B2      ; are in DPX+B0
                                   rrcf    DPX+B1
                                  else
0167 1A19                          rlcf    AARG+B1,W   ; rotate sign into
0168 191F                          rrcf    DPX+B3
0169 191E                          rrcf    DPX+B2
016A 191D                          rrcf    DPX+B1
016B 191C                          rrcf    DPX+B0
                                     fi
0009                               i = i+1
                                 if i < 8                ;test low byte
                                   btfss   BARG+B0,i
                                 else                    ; test high byte
016C 911B                          btfss   BARG+B1,i-8
                                 fi
016D C172                          goto    noadd9
                     add9
016E 6018                          movfp   AARG+B0,WREG
```


© 1993 Microchip Technology Inc.

```
016F 0F1E                           addwf   DPX+B2     ;add lsb
0170 6019                           movfp   AARG+B1,WREG
0171 111F                           addwfc  DPX+B3     ;add msb
                    noadd9
                                  if i < 8
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0172 1A19                           rlcf    AARG+B1,W  ; rotate sign into
0173 191F                           rrcf    DPX+B3
0174 191E                           rrcf    DPX+B2
0175 191D                           rrcf    DPX+B1
0176 191C                           rrcf    DPX+B0
                                    fi
000A                                i = i+1
                                  if i < 8             ;test low byte
                                    btfss   BARG+B0,i
                                  else                 ; test high byte
0177 921B                           btfss   BARG+B1,i-8
                                  fi
0178 C17D                           goto    noadd10
                    add10
0179 6018                           movfp   AARG+B0,WREG
017A 0F1E                           addwf   DPX+B2     ;add lsb
017B 6019                           movfp   AARG+B1,WREG
017C 111F                           addwfc  DPX+B3     ;add msb
                    noadd10
                                  if i < 8
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
017D 1A19                           rlcf    AARG+B1,W  ; rotate sign into
017E 191F                           rrcf    DPX+B3
017F 191E                           rrcf    DPX+B2
0180 191D                           rrcf    DPX+B1
0181 191C                           rrcf    DPX+B0
                                    fi
000B                                i = i+1
                                  if i < 8
                                    ;test low byte
                                    btfss   BARG+B0,i
                                  else
                                                       ; test high byte
0182 931B                           btfss   BARG+B1,i-8
                                  fi
0183 C188                           goto    noadd11
                    add11
0184 6018                           movfp   AARG+B0,WREG
0185 0F1E                           addwf   DPX+B2     ;add lsb
0186 6019                           movfp   AARG+B1,WREG
0187 111F                           addwfc  DPX+B3     ;add msb
                    noadd11
                                  if i < 8
                                    rlcf    AARG+B1,W  ; rotate sign into carry bit
                                    rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                    rrcf    DPX+B2     ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0188 1A19                           rlcf    AARG+B1,W  ; rotate sign into
0189 191F                           rrcf    DPX+B3
018A 191E                           rrcf    DPX+B2
018B 191D                           rrcf    DPX+B1
018C 191C                           rrcf    DPX+B0
                                    fi
```


© 1993 Microchip Technology Inc.

```
000C                                    i = i+1
                                      if i < 8
                                        ;test low byte
                                        btfss   BARG+B0,i
                                      else                    ; test high byte
018D 941B                               btfss   BARG+B1,i-8
                                      fi
018E C193                               goto    noadd12
                        add12
018F 6018                               movfp   AARG+B0,WREG
0190 0F1E                               addwf   DPX+B2      ;add lsb
0191 6019                               movfp   AARG+B1,WREG
0192 111F                               addwfc  DPX+B3      ;add msb
                        noadd12
                                      if i < 8
                                        rlcf    AARG+B1,W  ; rotate sign into carry bit
                                        rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                        rrcf    DPX+B2     ; are in DPX+B0
                                        rrcf    DPX+B1
                                       else
0193 1A19                               rlcf    AARG+B1,W  ; rotate sign into
0194 191F                               rrcf    DPX+B3
0195 191E                               rrcf    DPX+B2
0196 191D                               rrcf    DPX+B1
0197 191C                               rrcf    DPX+B0
                                          fi
000D                                    i = i+1
                                      if i < 8                ;test low byte
                                        btfss   BARG+B0,i
                                      else                                          ;
                                         test high byte
0198 951B                               btfss   BARG+B1,i-8
                                      fi
0199 C19E                               goto    noadd13
                        add13
019A 6018                               movfp   AARG+B0,WREG
019B 0F1E                               addwf   DPX+B2      ;add lsb
019C 6019                               movfp   AARG+B1,WREG
019D 111F                               addwfc  DPX+B3      ;add msb
                        noadd13
                                      if i < 8
                                        rlcf    AARG+B1,W  ; rotate sign into carry bit
                                        rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                        rrcf    DPX+B2     ; are in DPX+B0
                                        rrcf    DPX+B1
                                      else
019E 1A19                               rlcf    AARG+B1,W  ; rotate sign into
019F 191F                               rrcf    DPX+B3
01A0 191E                               rrcf    DPX+B2
01A1 191D                               rrcf    DPX+B1
01A2 191C                               rrcf    DPX+B0
                                          fi
000E                                    i = i+1
                                      if i < 8                ;test low byte
                                        btfss   BARG+B0,i
                                      else                    ; test high byte
01A3 961B                               btfss   BARG+B1,i-8
                                      fi
01A4 C1A9                               goto    noadd14
                        add14
01A5 6018                               movfp   AARG+B0,WREG
01A6 0F1E                               addwf   DPX+B2      ;add lsb
01A7 6019                               movfp   AARG+B1,WREG
01A8 111F                               addwfc  DPX+B3      ;add msb
                        noadd14
                                      if i < 8
                                        rlcf    AARG+B1,W  ; rotate sign into carry bit
                                        rrcf    DPX+B3     ; for i < 8, no meaningful bits
                                        rrcf    DPX+B2     ; are in DPX+B0
```


© 1993 Microchip Technology Inc.

```
                                    rrcf    DPX+B1
                                  else
01A9 1A19                           rlcf    AARG+B1,W  ; rotate sign into
01AA 191F                           rrcf    DPX+B3
01AB 191E                           rrcf    DPX+B2
01AC 191D                           rrcf    DPX+B1
01AD 191C                           rrcf    DPX+B0
                                    fi
000F                                i = i+1
                                if  SIGNED
01AE 1A19                           rlcf    AARG+B1,W  ; since BARG is alw
01AF 191F                           rrcf    DPX+B3     ; the last bit is k
01B0 191E                           rrcf    DPX+B2
01B1 191D                           rrcf    DPX+B1
01B2 191C                           rrcf    DPX+B0
                                endif
01B3 0002                      return
                    ;
                    ;*************************************************************
                         END

Errors   :     0
Warnings :     0
```

4

© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

MICROCHIP

AN532

Servo Control of a DC-Brush Motor

Author: Tim Bucella, Teknic Inc.

INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC-like features, the PIC17C42 offers excellent computation speed needed for real time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 µS (@16 MHz) allowing control loop sample times in the 2 KHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals of the PIC17C42 allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs. However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems offering a number of advantages:

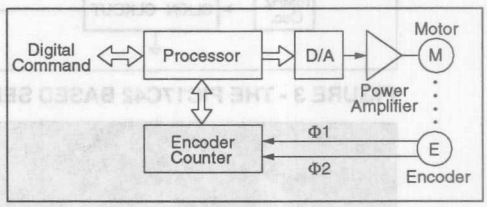
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of digital-to-analog converter, and a power amplifier, which delivers current or voltage to the motor.

FIGURE 1 - A TYPICAL SERVO SYSTEM



The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or the simple PID algorithm (implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The digital-to-analog conversion can be handled by a conventional DAC or by using pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output.

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because of its simplicity and reduced cost.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of Φ1 and Φ2 represents an increment of position. The phase of the signals is used to determine direction of rotation.

4

© 1993 Microchip Technology Inc.

FIGURE 2 - THE PIC17C42 SERVO SYSTEM

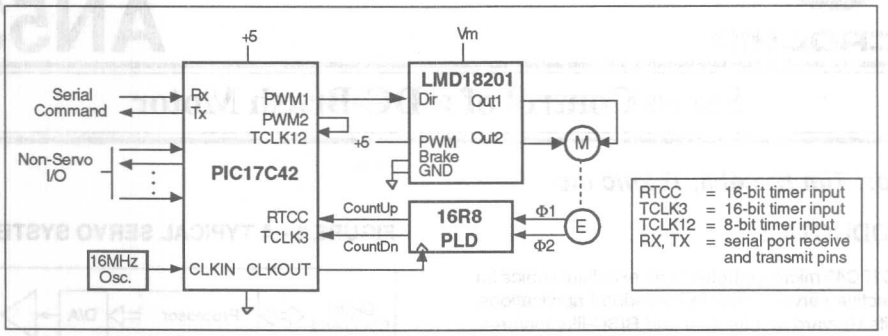


FIGURE 3 - THE PIC17C42 BASED SERVO CONTROL BOARD

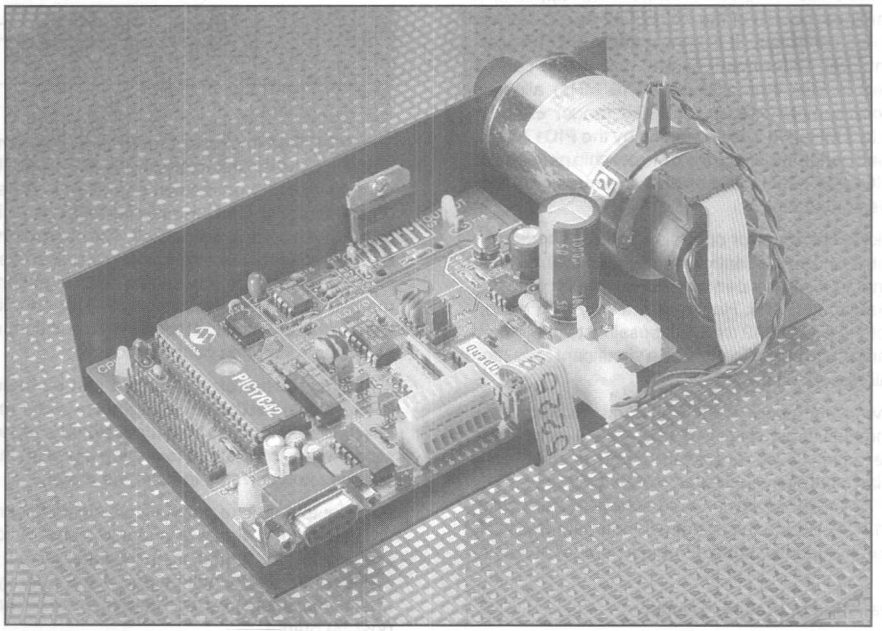
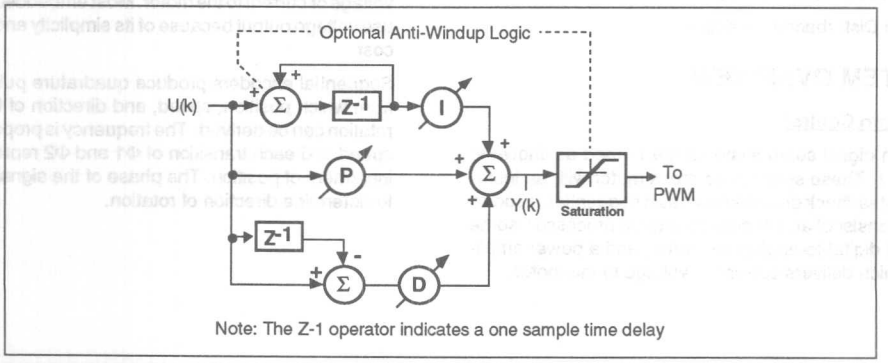


FIGURE 4 - DIGITAL PID IMPLEMENTATION



© 1993 Microchip Technology Inc.

These encoder signals are usually decoded using a small state machine into Count Up and Count Down pulses. These pulses are then routed to an N-bit, up-down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 3. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the digital-to-analog conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLD. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figures 5 and 6).

THE COMPENSATOR

PID is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, however it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 4. U(k) is the position or velocity error and Y(k) is the output.

This algorithm has been implemented using the PIC17C42 math library. Only 800 instruction cycles are required resulting in a 0.2mS PID execution time at 16 MHz.

Integrator wind-up is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown above avoids this problem by stopping the action of the integrator during output saturation.

MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5nS (at 16 MHz). This translates into 10-bit resolution at a 15.6KHz rate or 1 part in 800 (9 1/2-bit) resolution at 20KHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5mS or more and a mechanical time constant of 20.0mS or more. A 15KHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 2, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, V_m. In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 2. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 5). These signals are then fed to two 16-bit timers of the PIC17C42 (TMR3 and RTCC). A logic description for the PLD decoder is shown in Appendix B.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits deep, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. **For example, at a servo sample time of 1mS, the maximum encoder rate would be 3.2767 MHz.**

Counter wrap-around is not a concern because only the *difference* between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.

© 1993 Microchip Technology Inc.

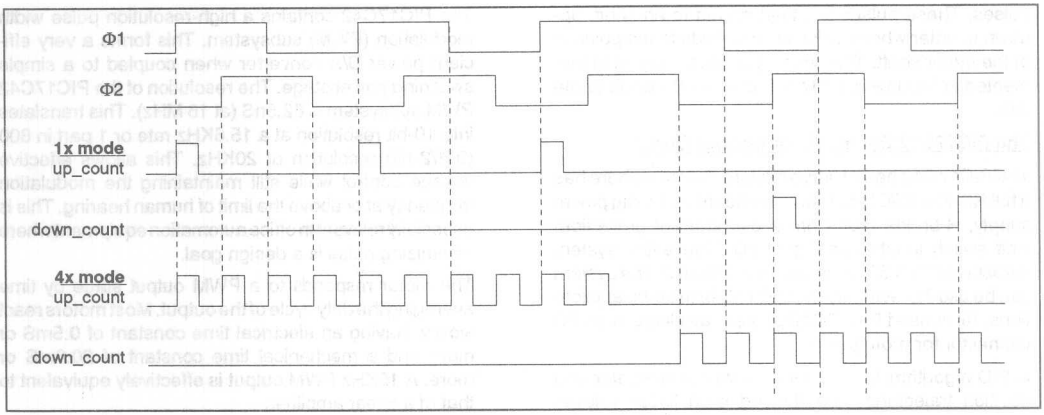
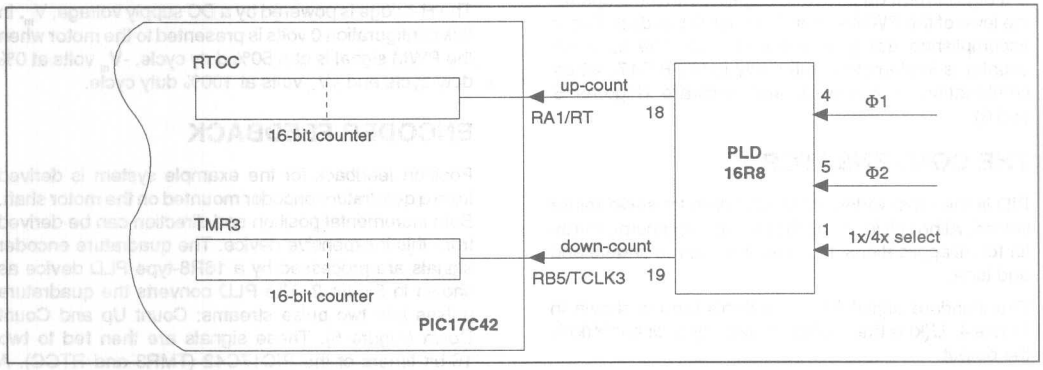


FIGURE 6 - ENCODER INTERFACE SCHEME



TRAJECTORY GENERATION

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular short move where maximum velocity was not reached (Figure 8).

The doPreMove subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The doMove routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

$V_K = V_{K-1} + A$ (A = Acceleration)

$P_K = P_{K-1} + V_{K-1} + A/2$

For more details on trajectory generation, see Appendix E.

IMPLEMENTATION DETAILS

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication and any exception processing (i.e. limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM, all at a constant, predefined rate. The ISR is initiated by a hardware timer (TMR2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N=16 (Figure 7).

© 1993 Microchip Technology Inc.

FIGURE 7 - SAMPLING SCHEME

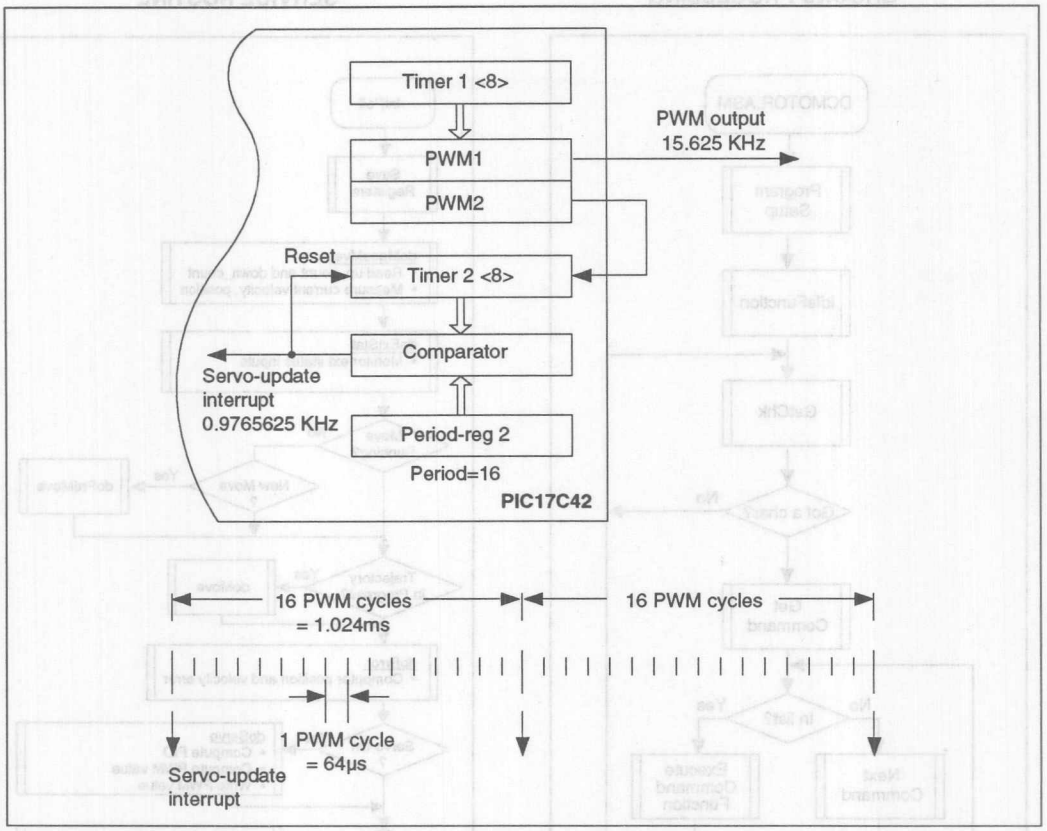
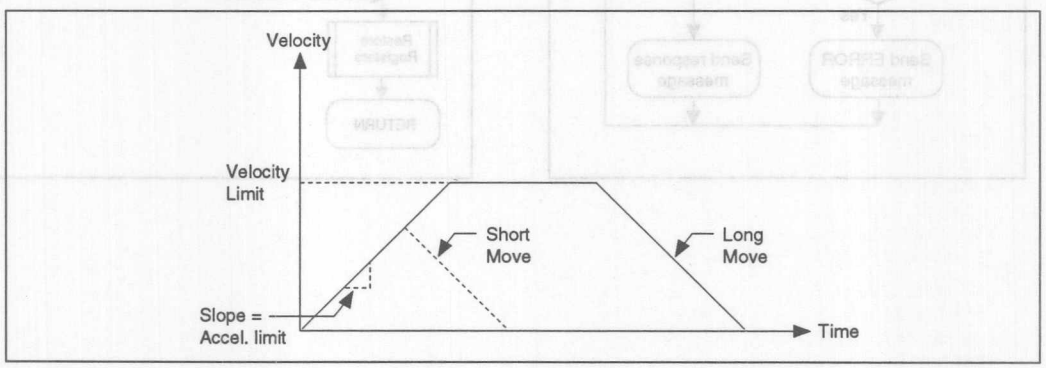


FIGURE 8 - VELOCITY RAMP SEGMENTS FOR POSITION MOVES



© 1993 Microchip Technology Inc.

FIGURE 9 - FLOWCHART FOR FORE-GROUND PROCESSING

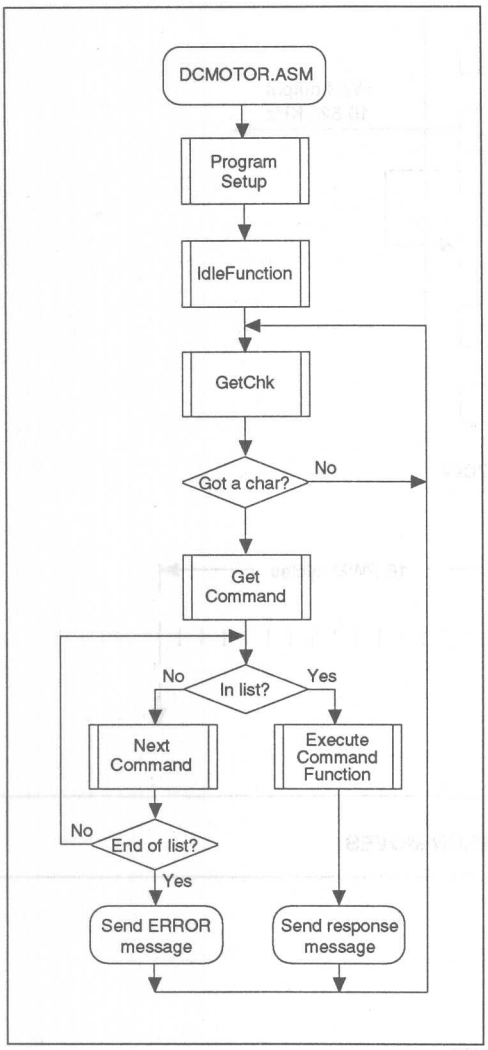
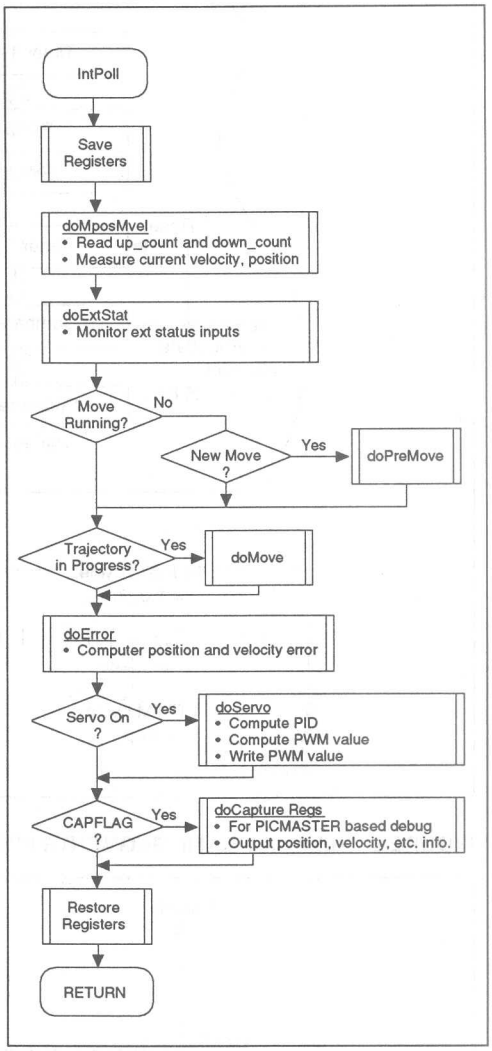


FIGURE 10 - FLOWCHART FOR INTERRUPT SERVICE ROUTINE



© 1993 Microchip Technology Inc.

The following events must occur in the interrupt service routine:

- Read Timers (RTCC & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error: U(k) = Reference Position - Current Position
- Calculate Y(k) using PID
- Set PWM output
- Manage other housekeeping tasks (i.e. service serial characters)

The entire ISR requires only 0.250mS to execute with 16MHz processor clock frequency.

COMMAND INTERFACE

The following commands are implemented and recognized by the user interface in the foreground loop.

Move (Value): M, [$-8,388,608_{10}$ to $8,388,607_{10}$]

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

Mode: O, (Type), [P,V, T]

An argument of "P" will cause all subsequent move commands to be incremental position moves. A "V" argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

Set Parameter: S, (#,Value) [00_h to FF_h, $-8,388,608_{10}$ to $8,388,607_{10}$]

Sets controller parameters to the value given. Parameters are shown in Table 1.

TABLE 1 - PARAMETERS

Parameter	$\#_h$	Range
Velocity Limit	00	0 to $8,388,607_{10}$ *
Acceleration Limit	01	0 to $8,388,607_{10}$ **
Kp: Proportional Gain	02	-32768_{10} to 32767_{10}
Kd: Differential Gain	03	-32768_{10} to 32767_{10}
Ki: Integral Gain	04	-32768_{10} to 32767_{10}

* (counts per sample time multiplied by 256)

** (counts per sample time per sample time multiplied by 256)

Read Parameter: R, (#) [00_h to FF_h]

Returns the present value of a parameter.

Shutter: C

Returns the time (in sample time counts 0 to $65,536_{10}$) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

Read commanded position: P

Returns the commanded position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read commanded velocity: V

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$

Read actual position: p

Returns the actual position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read actual velocity: v

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

External Status: X

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

TABLE 2 - EXTERNAL STATUS REGISTER BITS

Bit 7	index marker detected
Bit 6	+limit reached
Bit 5	-limit reached
Bit 4	input true
Bit 3-0	n/a

Move Status: Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

TABLE 3 - MOVE STATUS REGISTER BITS

Bit 7	move buffer empty
Bit 6	move complete
Bit 5-0	n/a

4

© 1993 Microchip Technology Inc.

Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,608$_{10}$ to 8,388,607$_{10}$]

Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.

Reset: Z

Performs a software reset.

Capture Servo-Response: c (#Count)

The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the doCaptureRegs procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see doCaptureRegs procedure). This is useful for debug purposes.

Disable Servo: s

This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.

Examples:

```
Z              ;Reset software (No <CR> required)
OV             ;Set velocity servo mode (No <CR>
                        ;required)
M 1000<CR>     ;Set velocity to 1000
M-1000<CR>     ;Set velocity to 1000 in reverse
                        ;direction
```

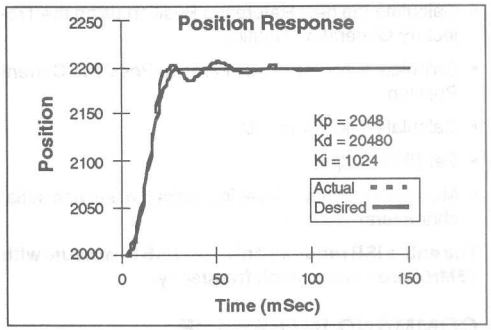
OPTIMIZING THE SYSTEM

Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER™ In-Circuit Emulator for the PIC17C42.

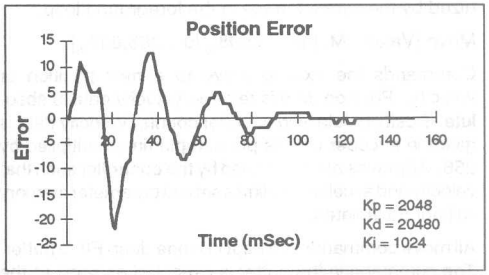
The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, 8K deep trace buffer and external logic probes. It's user interface software runs under Windows™ 3.1 with pull-down menus and on-line help. The PICMASTER software also support dynamic data exchange (DDE) through which it is possible to send its trace buffer information to a spreadsheet, such as EXCEL™, also running under windows.

To tune the PID, first a small amount of diagnostics code is added in the servo routine (doCaptureRegs). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using TABLWT instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle TABLWT instructions. Next, the trace buffer is transferred to EXCEL and the various

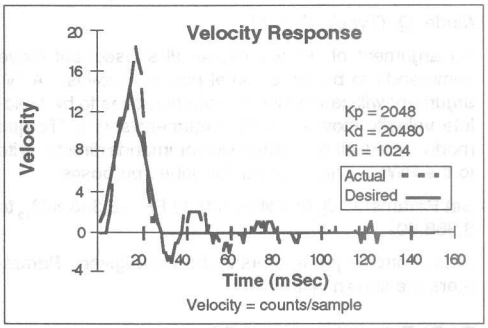
DESIRED/ACTUAL POSITION



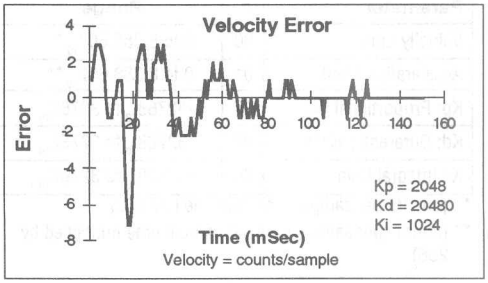
POSITION ERROR



DESIRED/ACTUAL VELOCITY



VELOCITY ERROR



© 1993 Microchip Technology Inc.

parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering Kp, Ki and Kd, and plotting the results, the system can be fine tuned.

Under windows multitasking environment, using PICMASTER emulator this can be done in real time as described below.

Three sessions are set up under windows:

1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are setup to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setup have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

TABLE 4 - SERVO CONTROL CHIP COMPARISON

	LM629 @8MHz	PIC17C42 @16MHz	PIC17C42 @25MHz
Max Encoder Rate	1MHz	3.3 MHz	4.5 MHz
Servo Update Time	-	0.25 ms	0.16 ms
Max Sampling Frequency	4 KHz	2-3 KHz	4-5 KHz

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.

Credit

This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

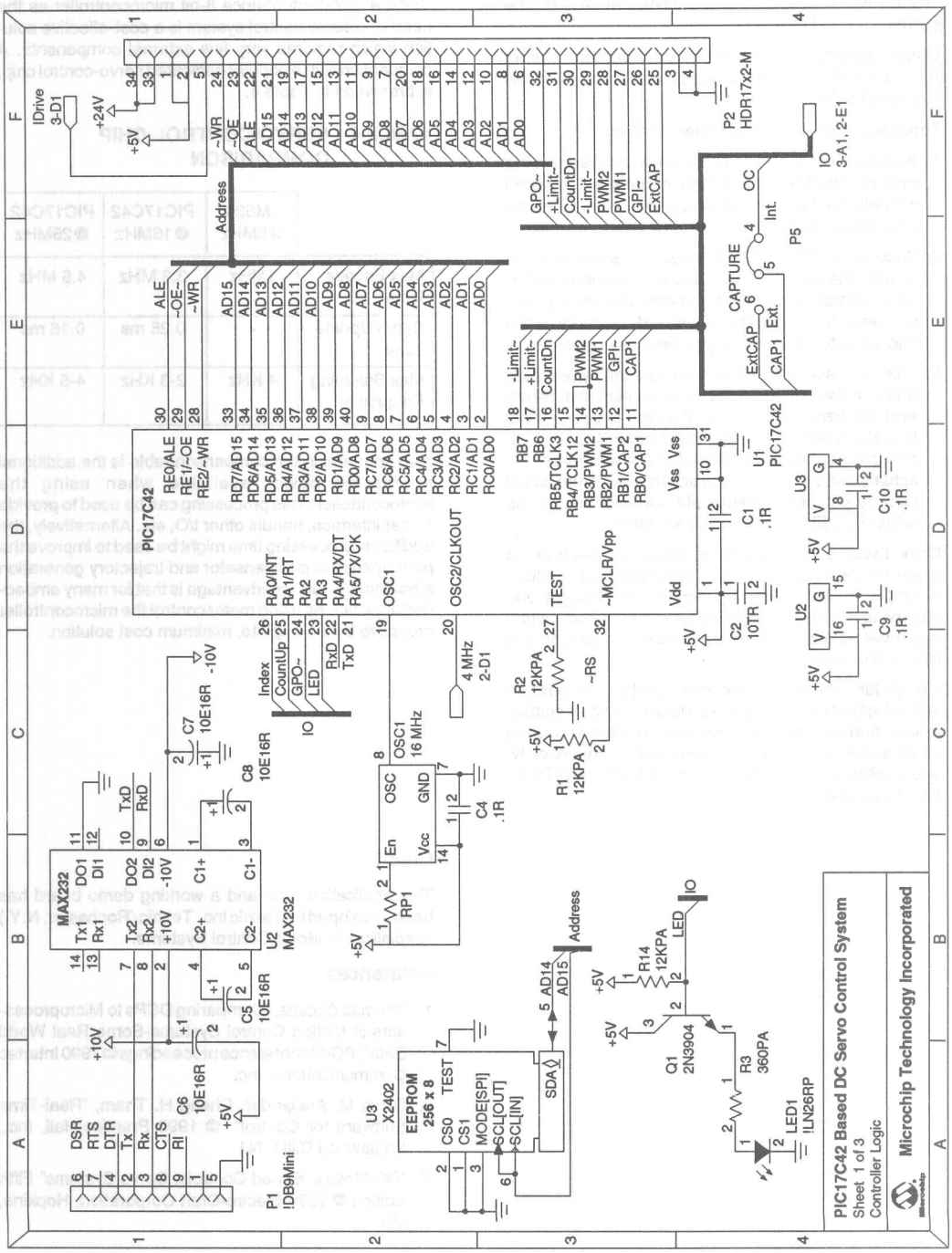
References

1. Thomas Bucella, "Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data", PCIM conference proceedings © 1990 Intertec Communications, Inc.
2. David M. Auslander, Cheng H. Tham, "Real-Time Software for Control" © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ
3. "DC Motors, Speed Controls, Servo Systems" Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

Windows is a trademark of Microsoft Corporation.

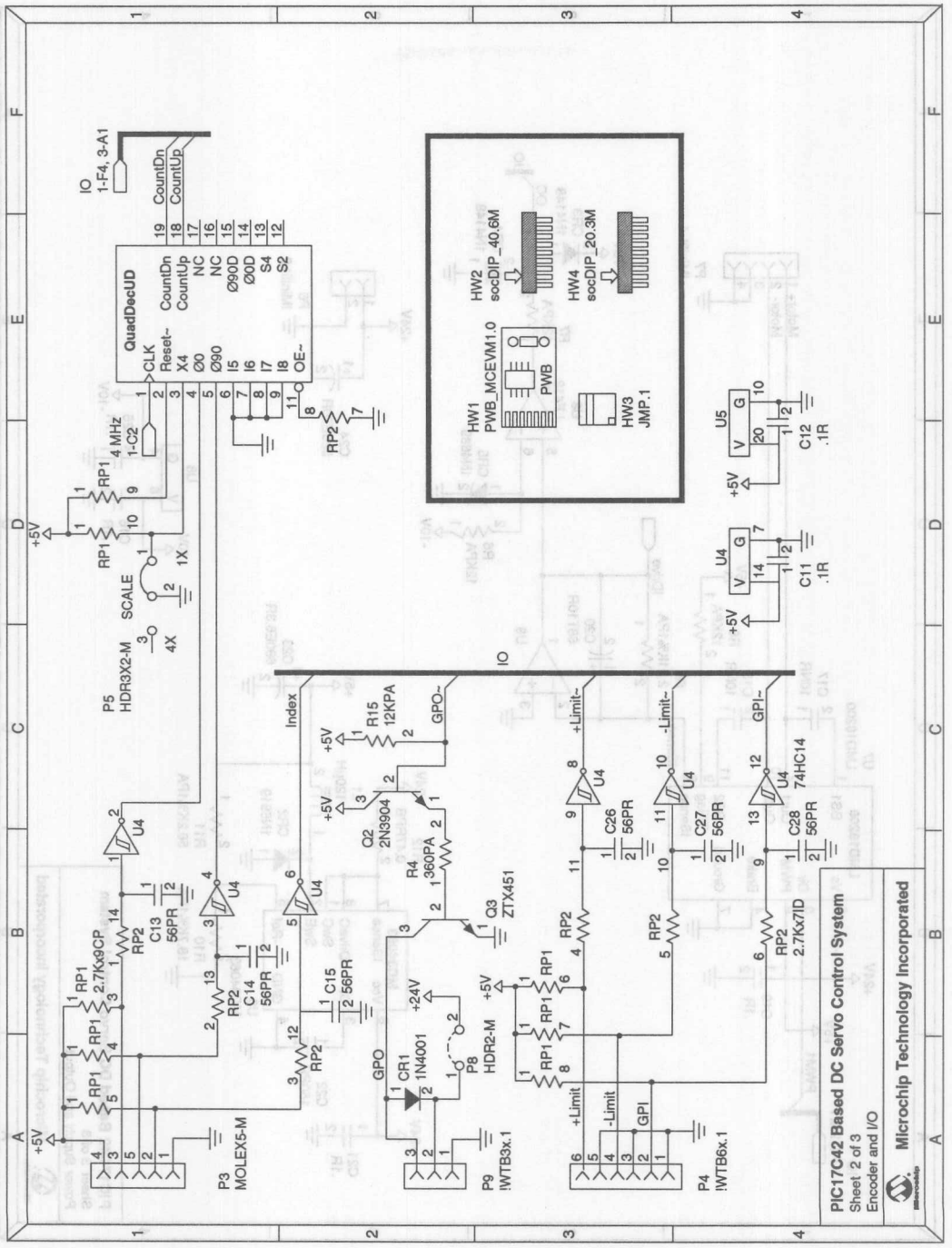
© 1993 Microchip Technology Inc.

APPENDIX A: SCHEMATIC DIAGRAM

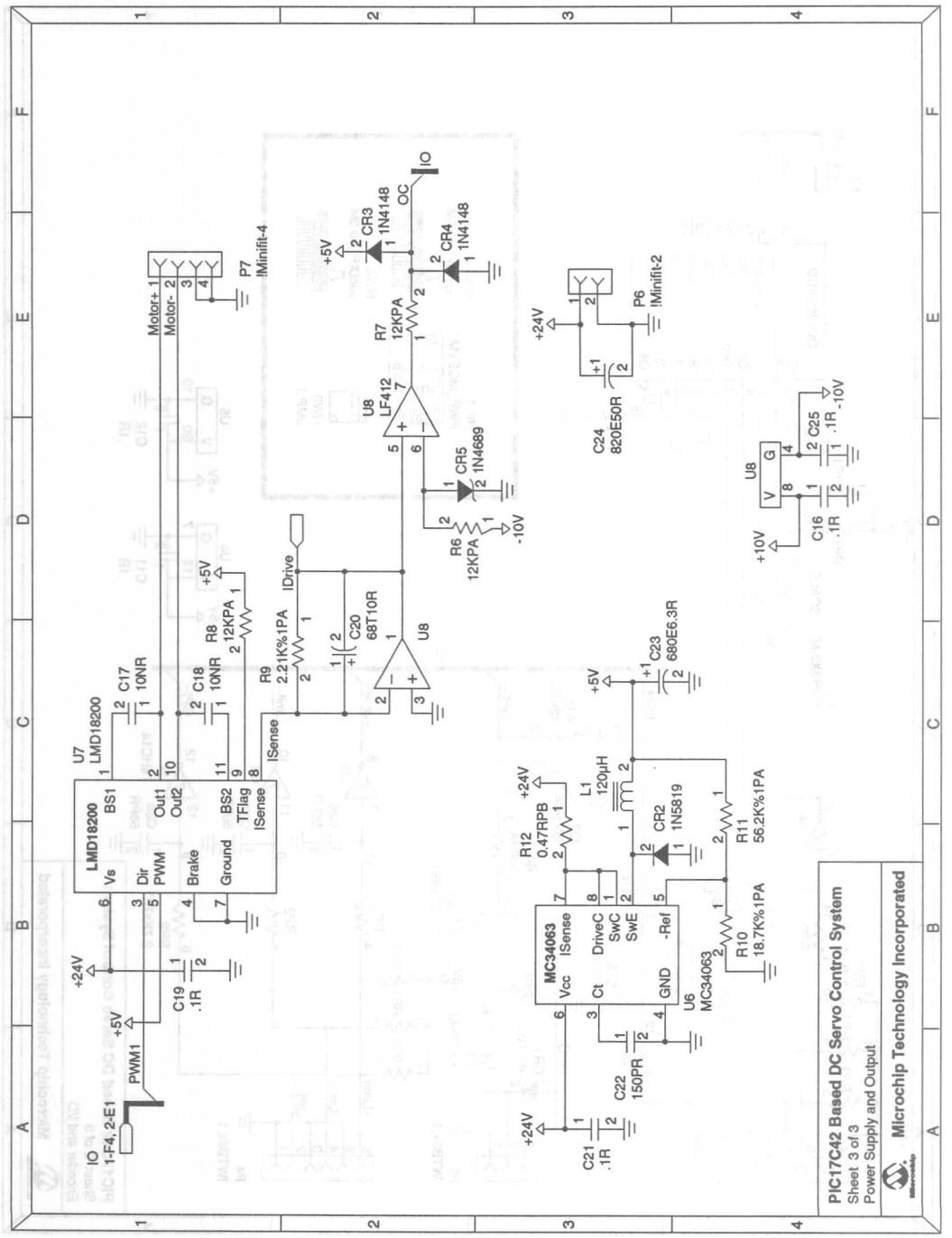


© 1993 Microchip Technology Inc.

APPENDIX A (CONT.): SCHEMATIC DIAGRAM



APPENDIX A (CONT.): SCHEMATIC DIAGRAM



© 1993 Microchip Technology Inc.

APPENDIX B: ENCODER PLD EQUATIONS

```
Combination quadrature decoder and input synchronizer.  This design
allows 1x decoding or 4x decoding based on the X4 pin.

        * Ver 1.0 - November 8, 1991
}
MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

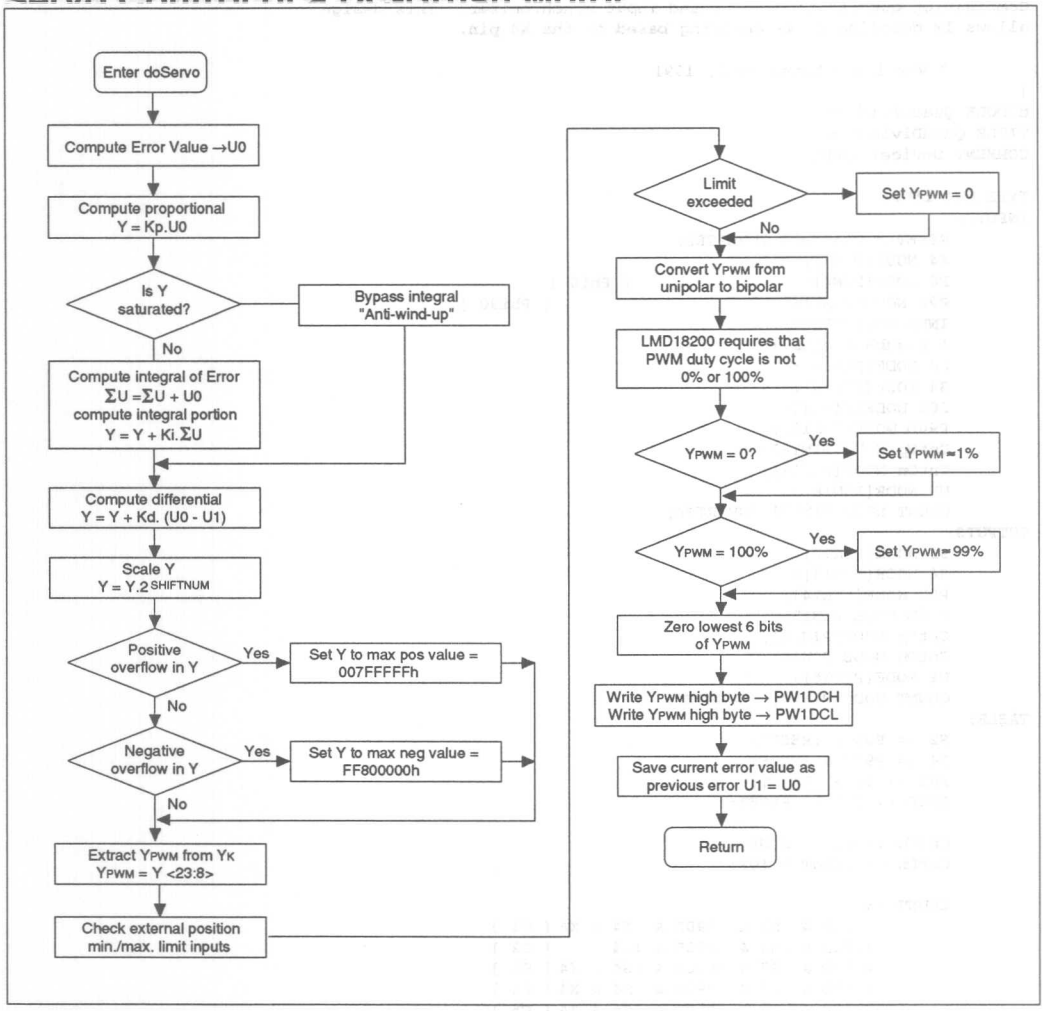
TYPE MMI 16R8;
INPUTS;
        RESET NODE[PIN2] INVERTED;
        X4 NODE[PIN3];
        P0 NODE[PIN4];                  { Phi0 }
        P90 NODE[PIN5];                 { Phi90 }
        INDX NODE[PIN6];
        { Feedback pins }
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
OUTPUTS;
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
TABLE;
        S2 := P0D & !RESET;
        S4 := P90D & !RESET;
        P0D := P0 & !RESET;
        P90D := P90 & !RESET;

        CntUp := COUNT & UP;
        CntDn := COUNT & !UP;

        COUNT :=
              (  P0D &  S2 & !P90D &  S4 & X4 { C1 }
               +!P0D & !S2 &  P90D & !S4      { C2 }
               +!P0D &  S2 & !P90D & !S4 & X4 { C3 }
               + P0D & !S2 &  P90D &  S4 & X4 { C4 }
               + P0D &  S2 &  P90D & !S4 & X4 { C5 }
               +!P0D & !S2 & !P90D &  S4      { C6 }
               +!P0D &  S2 &  P90D &  S4 & X4 { C7 }
               + P0D & !S2 & !P90D & !S4 & X4 { C8 }
               ) & !RESET;
        UP :=
               (
                !P0D &  S2 & !P90D &  S4
               +!P0D &  S2 &  P90D &  S4
               +!P0D &  S2 &  P90D & !S4
               + P0D &  S2 &  P90D & !S4
               + P0D & !S2 &  P90D & !S4
               + P0D & !S2 & !P90D & !S4
               + P0D & !S2 & !P90D &  S4
               +!P0D & !S2 & !P90D &  S4
               ) & !RESET;
END;

END QuadDivider;
```

© 1993 Microchip Technology Inc.



© 1993 Microchip Technology Inc.

APPENDIX C: (PART 2): PID ALGORITHM CODE LISTING

```
;*************************************************************************
; NAME:         doServo
;
; DESCRIPTION: Performs the servo loop calculations.
;
doServo
        MOV16   POSERROR,U0             ; save new position error in U0

        LOADAB  U0,KP                   ; compute KP*U0                    Basic PID
        CALL    Dmult                                                      calculation
        MVPF32  DPX,Y                   ; Y=KP*U0

        CLRF    WREG                    ; if previous output saturated, do anti-
        CPFSGT  SATFLAG                 ; not accumulate integrator       wind-
        CALL    doIntegral                                                 up

        LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL
        CALL    Dmult
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL

        MVFP16  U0,AARG                 ; compute KV*(U0-U1)
        SUB16   U1,AARG
        MVFP16  KV,BARG
        CALL    Dmult
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)

        CLRF    WREG                                                       Scale Y
        CPFSGT  SHIFTNUM                    ; scale Y by SHIFTNUM
        GOTO    grabok                  ; Y = Y * (2**SHIFTNUM)
        MOVFP   SHIFTNUM,TMP
grabloop
        RLC32   Y
        DECFSZ  TMP
        GOTO    grabloop

grabok
        CLRF    SATFLAG
        BTFSC   Y+B3,MSB                ; saturate to middle 16 bits,
        GOTO    negs                    ; keeping top 10 bits for PW1DCH
poss                                    ; and PW1DCL
        MOVFP   Y+B2,WREG               ; check if Y >= 2**23              If positive
        ANDLW   0x80                                                       overflow, saturate
        IORWF   Y+B3                                                       y to maximum
        CLRF    WREG                                                       positive number
        CPFSGT  Y+B3
        GOTO    zero6bits               ; if not, zero 6 bits

        INCF    SATFLAG                 ; if so, set Y=0x007FFFFF
        CLRF    Y+B3                    ; clear for debug purposes
        MOVLW   0x7F
        MOVPF   WREG,Y+B2
        SETF    Y+B1
        SETF    Y+B0
        GOTO    zero6bits
negs
        MOVFP   Y+B2,WREG               ; check if Y <= -2**23             If negative
        IORLW   0x7F                                                       overflow, saturate
        ANDWF   Y+B3                                                       y to maximum
        SETF    WREG                                                       negative value
        CPFSLT  Y+B3
        GOTO    zero6bits               ; if not, zero 6 bits

        SETF    SATFLAG                 ; if so, set Y = 0xFF800000
        SETF    Y+B3
        CLRF    Y+B2
        BSF     Y+B2,MSB
        CLRF    Y+B1
        CLRF    Y+B0
```

4

© 1993 Microchip Technology Inc.

```
zero6bits
        MOV24   Y+B1,YPWM+B0            ; move Y to YPWM and zero 6 bits
doTorque                                ; entry point for torque mode
        MOVLW   0xC0
        ANDWF   YPWM+B0

        BTFSC   YPWM+B1,MSB
        GOTO    tmlimit
tplimit
        BTFSS   EXTSTAT,BIT6
        GOTO    mplimitok
        CLR32   YPWM
        GOTO    mplimitok
tmlimit
        BTFSS   EXTSTAT,BIT5
        GOTO    mplimitok
        CLR32   YPWM
mplimitok
        MOVLW   PW1DCH_INIT             ; adjustment from bipolar to unipolar
        MOVPF   WREG,TMP+B1             ; for 50% duty cycle
        MOVLW   PW1DCL_INIT
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

        CLRF    TMP+B1                  ; correct by 1 LSB
        MOVLW   0x40                    ; add one to bit5 of PW1DCL
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

testmax
        CLRF    TMP+B2                  ; check pwm maximum limit
        CLRF    YPWM+B2                 ; LMD18200 must have a minimum pulse
        CLRF    YPWM+B3                 ; so duty cycle must not be 0 or 100%
        MVFP16  YPWMAX,TMP
        SUB24   YPWM,TMP
        BTFSS   TMP+B2,MSB
        GOTO    testmin
        MOV16   YPWMAX,YPWM             ; saturate to max
        GOTO    limitok
testmin
        CLRF    TMP+B2                  ; check pwm minimum limit
        CLRF    YPWM+B2
        CLRF    YPWM+B3
        MVFP16  YPWMIN,TMP
        SUB24   YPWM,TMP
        BTFSC   TMP+B2,MSB
        GOTO    limitok
        MOV16   YPWMIN,YPWM             ; saturate to min
limitok
        MOVLB   BANK3                   ; set new duty cycle
        MOVFP   YPWM+B0,PW1DCL
        MOVFP   YPWM+B1,PW1DCH

        MOV16   U0,U1                   ; push errors into U(k-1)

        RETURN
;*****************************************************************************
```

If external position limits have been reached then zero PWM output.

Convert PWM from unipolar to bipolar

PWM cycle must not be 0% of 100%

Write PWM values to PWM registers

© 1993 Microchip Technology Inc.

APPENDIX D: ENCODER INTERFACE ROUTINE

```
;*******************************************************************************
; NAME:                    doMPosMVel
;
; DESCRIPTION:     Calculates current position from UpCount and DownCount
;
doMPosMVel

; Do UpCounter first

                    MVFP16  UPCOUNT,TMP+B0          ; save old upcount
readUp
                    MOVPF   RTCCH,WREG
                    MOVPF   RTCCL,UPCOUNT+B0
                    CPFSEQ  RTCCH                   ; Skip next if HI hasn't changed
                    GOTO    readUp                  ; HI changed, re-read LO
                    MOVPF   WREG,UPCOUNT+B1         ; OK to store HI now

                    CLRF    MVELOCITY+B0            ; clear bits below binary point

                    MOV16   UPCOUNT,MVELOCITY+B1    ; compute upcount increment
                    SUB16   TMP+B0,MVELOCITY+B1

; Now do DownCounter

                    MVFP16  DOWNCOUNT,TMP+B0        ; save old downcount
readDown
                    MOVLB   BANK2                   ; timers in Bank 2
                    MOVPF   TMR3H,WREG
                    MOVPF   TMR3L,DOWNCOUNT+B0
                    CPFSEQ  TMR3H                   ; Skip next if HI hasn't changed
                    GOTO    readDown                ; HI changed, re-read LO
                    MOVPF   WREG,DOWNCOUNT+B1       ; OK to store HI now

                    MVFP16  DOWNCOUNT+B0,TMP+B2     ; compute downcount increment
                    SUB16   TMP+B0,TMP+B2

                    SUB16   TMP+B2,MVELOCITY+B1     ; compute new measured velocity

                    CLRF    MVELOCITY+B3            ; sign extend measured velocity for
                    BTFSC   MVELOCITY+B2,MSB        ; 24 bit addition to measured posi-
tion
                    SETF    MVELOCITY+B3



                    ADD24   MVELOCITY+B1,MPOSITION; compute new measured position
                                                    ; delta position = measured velocity

                    RETURN

;*******************************************************************************
```

APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

doPreMove:

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if abs (MOVVAL) > 2, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and MOVVAL/2 in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at modeready, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

doMove:

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by

$$x(t) = x0 + v0*t + a*(t**2)/2, v(t) = v0 + a*t$$

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by

$$P(k) = P(k-1) + V(k-1) + A/2, \quad V(k) = V(k-1) + A,$$

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by

$$P(k-1) = P(k) - V(K-1) - A/2, \quad V(K-1) = V(k) - A.$$

In position mode, the actual shape of the velocity profile depends on the values of V, A and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speed down sections of the move. Phase 1 is defined as the section of the move where the commanded position is less than half the move, and phase 2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

FIGURE A - SPEED PROFILE FOR TRAPEZOIDAL MOVES

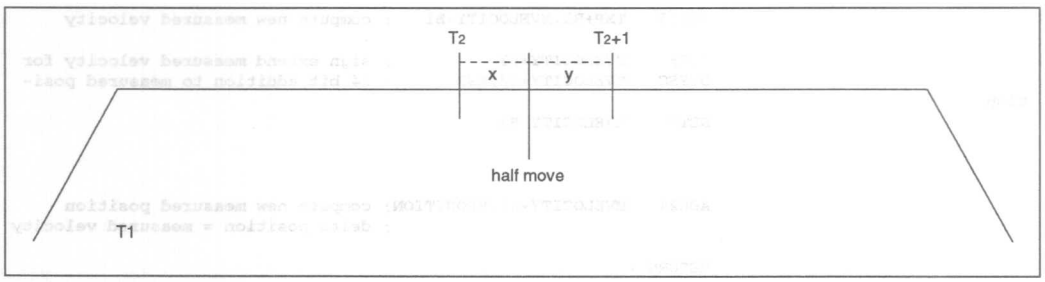


FIGURE B - SPEED PROFILE FOR TRIANGULAR MOVES

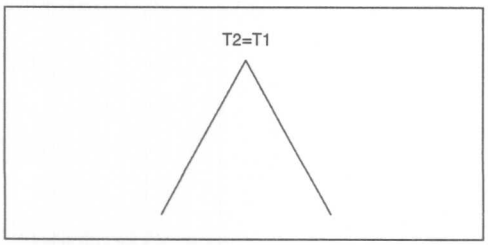
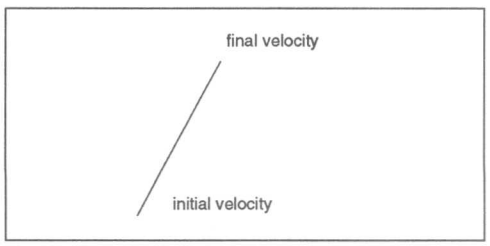


FIGURE C - SPEED PROFILE FOR VELOCITY MOVES



© 1993 Microchip Technology Inc.

Furthermore, let x be the amount of undershoot and y the amount of overshoot of half the move at T2. Discretization error is minimized by using the values of x and y whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by min. (2x, 2y), resulting in the condition that if 2x > 2y, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by min. (2x, y-x), yielding the condition that if 3x > y, take one more step during the flat section of phase2.

At the beginning of doMove, MOVTIME is incremented and doPosVel is called to evaluate the next proposed values of commanded position and velocity under the current value of A. In position mode, phase1, the original position plus half the move minus the new proposed commanded position is calculated and placed in MOVDEL, with the previous MOVDEL saved in MOVTMP. As half the move would be passed, MOVTMP = -x and MOVDEL = y, with y>0 for the first time indicating that phase1 is about to be completed. Therefore, if y<0, we continue in phase1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, reevaluate the iterative equations with the new acceleration, set T1=MOVTIME-1, and execute the move.

Since T1 is cleared in doPreMove, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that y>0, we drop into code that is executed only one time, with phase2 beginning on the next step. If T1=0, maximum velocity has not yet been reached, so T1=T2 and the velocity profile is triangular. In this case, A is negated for speed down, and if x>y, one more step is needed to minimize the discretization error. So A is negated, the proposed step undone, A is again negated for speed down and the step recalculated and executed, with T2=T1=MOVTIME-1.

If T1 is not zero, indicating that we are in the flat section of phase1, then go to t2net1, where T2=MOVTIME-1, and if 3x>y, then one more phase2 flat step is necessary to minimize the discretization error. PH2FLAT is defined as the number of steps in the flat section of phase2, and is used as a counter during its completion. If 3x>y, then PH2FLAT=T2-T1, otherwise PH2FLAT=T2-T1-1 and phase1 is finally complete. All subsequent steps will proceed through phase2, first deciding if the flat section is finished by checking if PH2FLAT has reached zero. If not, go to flat where PH2FLAT is decremented, and tested if zero. If so, the speed down section is begun by calculating the appropriate signed acceleration limit A, and executing the last of the flat section moves. For all following steps, PH2FLAT=0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speed down.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continued unless the final velocity is zero.

© 1993 Microchip Technology Inc.

APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

```
MPASM B0.54                                                    PAGE  1
"Revision:    2.0, 27 June 93"

                        SubTitle        "Revision:   2.0, 27 June 93"
                ;*************************************************************************
                ;
                ;       Revised: 8/5/92
                ;       CREDIT:  Developed by Teknic Inc. 1992
                ;
                ;  Assembled using MPASM. User's with ASM17 are suggested to get Microchip's
                ;  new universal assembler (MPASM). To assemble with ASM17, all "if", "else"
                ;  "endif" directives must be replaced by "#if", "#else" and "#endif"
                ;  respectively.
                ;*************************************************************************
                        PROCESSOR       PIC17C42
                        LIST            COLUMNS=120, XREF=YES, NOWRAP,LINES=255, R=DEC
                ;*************************************************************************
                ;
00F4 2400       MASTER_CLOCK    set     16000000        ; Input Clock Freq in Hz

03E8            _SAMPLE_RATE    set     1000            ; Sample rate in Hz
07D0            _ENCODER_RATE   set     2000            ; 2000 Pulses/rev
1770            _RATED_SPEED    set     6000            ; in RPM

2580            _BAUDRATE_      set     9600

                ;*************************************************************************

                        include "17c42.h"


                        include "17c42.mac"     ; General Purpose Macros


0058             #define _PICMASTER_DEBUG       TRUE    ; Enable PIC-MASTER TRACE Capture
0059             #define _SERVO_PID             TRUE    ; PID computation based on error
005A             #define DECIO                  TRUE    ; true for decimal, false for hex
005B             #define _SERIAL_IO             TRUE

                        include "dcmotor.h17"           ; Initialization, Global Defs,
                ;*************************************************************************
                ;
                ; Header file for dcmotor.asm:
                ; Revised: 8/5/92
                ;*************************************************************************
                ;
                ; hardware constants
                ;
                ;

005C             #define _SET_BAUD_RATE(bps)    ((10*MASTER_CLOCK/(64*bps))+5)/10 - 1
003D 0900       CLKOUT          set     MASTER_CLOCK >> 2   ; Clock Out = CLKIN/4


0006            TCON1_INIT      set     0x06
003F            TCON2_INIT      set     0x3F
00FF            PR1_INIT        set     0xFF                ; set pwm frequency to CLKOUT/256
```


© 1993 Microchip Technology Inc.

```
000F                    PR2_INIT        set     ((10*MASTER_CLOCK/(4*(PR1_INIT+1)*_SAMPLE_RATE))+5)/
                        ;pw1dcH_INIT    set     (((PR1_INIT+1) << 8)) >> 9
                        ;pw1dcL_INIT    set     (((((PR1_INIT+1) << 8)) >> 1) & 0xff)

007F                    PW1DCH_INIT     set     0x7F
00C0                    PW1DCL_INIT     set     0xC0

0080                    RTCSTA_INIT     set     0x80
0090                    RCSTA_INIT      set     0x90
0020                    TXSTA_INIT      set     0x20
0019                    SPBRG_INIT      set     _SET_BAUD_RATE(_BAUDRATE_)

00F3                    DDRB_INIT       set     0xF3
0000                    DDRD_INIT       set     0x00
                        ;
                        ;
                        ;       max and min pwm values
                        ;
0040                    PWMINL          set     0x40
0001                    PWMINH          set     0x01                    ; 0x0000 + 0x0140 (min 10 bit
0080                    PWMAXL          set     0x80
00FE                    PWMAXH          set     0xFE                    ; 0xFFC0 - 0x0140 ( max 10 bit
                        ;
                        ;
                        ;*****************************************************************************
                        ;       global  variables
                        ;
                        CBLOCK  0x18
0018 0004                       DPX,DPX1,DPX2,DPX3                      ; arithmetic accumula
001C 0004                       AARG,AARG1,BARG,BARG1                   ; multiply arguments
                        ENDC

                        CBLOCK  0x18
0018 0004                       TMP,TMP1,TMP2,TMP3                      ; temporary variables
001C 0004                       MOVTMP,MOVTMP1,MOVTMP2,MOVTMP3          ; move temporary
                        ENDC

                        CBLOCK  0x20
0020 0003                       VL,VL1,VL2                              ; velocity limit
0023 0003                       AL,AL1,AL2                              ; acceleration limit
0026 0000
0026 0002                       KP,KP1                                  ; proportional gain
0028 0002                       KV,KV1                                  ; velocity gain
002A 0002                       KI,KI1                                  ; integral gain
002C 0000
002C 0001                       IM                                      ; integrator mode
002D 0002                       FV,FV1                                  ; velocity feedforward
002F 0002                       FA,FA1                                  ; acceleration
0031 0000
0031 0003                       VALBUF,VALBUF1,VALBUF2                  ; iovalue buffer
0034 0000
0034 0003                       DVALBUF,DVALBUF1,DVALBUF2               ; iovalue buffer
0037 0002                       ISRBSR,ISRWREG                          ; isr save storage
0039 0004                       CMDCHAR,CMDTEMP,CMDPTRH,CMDPTRL         ; command interface
003D 0003                       PARTEMP,PARLEN,PARPTR                   ; parameter variables
0040 0000
0040 0003                       CPOSITION,CPOSITION1,CPOSITION2         ; shutter commanded
0043 0003                       CVELOCITY,CVELOCITY1,CVELOCITY2         ; shutter commanded
0046 0003                       CMPOSITION,CMPOSITION1,CMPOSITION2      ; shutter measured
0049 0003                       CMVELOCITY,CMVELOCITY1,CMVELOCITY2      ; shutter measured
004C 0000
004C 0002                       STRVALH,STRVALL                         ; string io variables
004E 0003                       HEXVAL,HEXTMP,HEXSTAT                   ; hex io variables
0051 0000
0051 0002                       OPOSITION,OPOSITION1
0053 0002                       OPOSITION2,OPOSITION3                   ; original commanded
0055 0003                       POSITION,POSITION1,POSITION2            ; commanded position
0058 0003                       VELOCITY,VELOCITY1,VELOCITY2            ; commanded velocity
005B 0000
```

© 1993 Microchip Technology Inc.

```
005F 0004                       MOVVAL,MOVVAL1,MOVVAL2,MOVVAL3        ; move value
0063 0004                       HMOVVAL,HMOVVAL1,HMOVVAL2,HMOVVAL3    ; half move value
0067 0002                       MOVTIME,MOVTIME1                      ; move time in sample
0069 0000
0069 0001                       MOVSIGN                               ; 0x00 for positive,
0x80 for
006A 0002                       T1,T11                                ; time to maximum
006C 0002                       T2,T21                                ; time for half the
006E 0002                       TAU,TAU1                              ; total move time
0070 0001                       NMODE                                 ; next move modetype
0071 0001                       MODE                                  ; move modetype
0072 0000
0072 0003                       MPOSITION,MPOSITION1,MPOSITION2       ; measured position
0075 0002                       MVELOCITY,MVELOCITY1
0077 0002                       MVELOCITY2,MVELOCITY3                 ; measured velocity
0079 0003                       POSERROR,POSERROR1,POSERROR2          ; position error
007C 0003                       VELERROR,VELERROR1,VELERROR2          ; velocity error
007F 0000
007F 0001                       SIGN                                  ; multiply sign
0080 0000
0080 0004                       Y,Y1,Y2,Y3                            ; Y(k) before pwm
0084 0004                       U0,U01,U1,U11                         ; saturated error at
0088 0000
0088 0004                       YPWM,YPWM1,YPWM2,YPWM3                ; pwm input
008C 0004                       YPWMIN,YPWMIN1,YPWMAX,YPWMAX1         ; pwm input limits
0090 0000
0090 0001                       SERVOFLAG                             ; servoflag = 0 => no
0091 0001                       MODETYPE                              ; mode
0092 0001                       EXTSTAT                               ; external status
0093 0001                       MOVSTAT                               ; move status register
0094 0001                       MOVFLAG                               ; move flag
0095 0001                       SATFLAG                               ; saturation flag
0096 0002                       INTEGRAL,INTEGRAL1                    ; integrator
0098 0000
0098 0004                       DECVAL,DECSTAT,DECTMP,DECSIGN         ; decimal io variables
009C 0000
009C 0004                       A,A1,A2,A3                            ; commanded accelera
00A0 0004                       V,V1,V2,V3                            ; commanded velocity =
00A4 0004                       MOVPBUF,MOVPBUF1,MOVPBUF2,MOVPBUF3    ; commanded position
00A8 0004                       MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3    ; commanded velocity
00AC 0000
00AC 0002                       UPCOUNT,UPCOUNT1                      ; running up counter
00AE 0002                       DOWNCOUNT,DOWNCOUNT1                  ; running down counter
00B0 0000
00B0 0004                       MOVDEL,MOVDEL1,MOVDEL2,MOVDEL3        ; move discretization
00B4 0002                       PH2FLAT,PH2FLAT1                      ; phase 2 flat itera
00B6 0003                       INDEXPOS,INDEXPOS1,INDEXPOS2          ; position at last
00B9 0000
00B9 0001                       SHIFTNUM                              ; # of bit shifts from
00BA 0000
00BA 0001               if      _PICMASTER_DEBUG
00BB 0000               ;*************************************************************************
00BB 0000               ;       For PICMASTER Debug/servo tuning Purposes Only
00BB 0000
00BB 0001                       CAPFLAG                               ; trace capture flag
00BC 0002                       CAPCOUNT,CAPCOUNT1                    ; PICMASTER trace
00BE 0002                       CAPTMP,CAPTMP1                        ; trace capture
00C0 0000
00C0 0000               ;*************************************************************************
```


© 1993 Microchip Technology Inc.

```
00C0 0001             endif
00C1 0000
00C1 0002                       ZERO,ONE                ; constants
00C3 0002                       tblptrlTemp,tblptrhTemp ; temp TABLE Pointers for
00C5 0002                       TblLatLo, TblLatHi      ; Table Latch for ISR save
00C7 0001                       alustaTemp              ; temp alusta
00C8 0000
                        ENDC

                    ;*************************************************************************
                    ; NAME:          AUTONO
                    ;
                    ; DESCRIPTION:   Sets no auto increment or decrement
                    ;
                    ; TIMING (cycles): 4

                    AUTONO MACRO

                                    BSF     _fs0
                                    BSF     _fs1
                                    BSF     _fs2
                                    BSF     _fs3

                    ENDM
                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:          AUTOINC
                    ;
                    ; DESCRIPTION:   Set auto increment
                    ;
                    ; TIMING (cycles): 4
                    ;

                    AUTOINC MACRO

                                    BSF     _fs0
                                    BCF     _fs1
                                    BSF     _fs2
                                    BCF     _fs3

                    ENDM
                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:          AUTODEC
                    ;
                    ; DESCRIPTION:   Sets auto decrement
                    ;
                    ; TIMING (cycles): 4
                    ;

                    AUTODEC MACRO

                                    BCF     _fs0
                                    BCF     _fs1
                                    BCF     _fs2
                                    BCF     _fs3

                    ENDM
                    ;*************************************************************************

                    ;*************************************************************************
```

4

```
                ; NAME:         LOADAB
                ;
                ; DESCRIPTION:  Loads extended math library AARG and BARG
                ;
                ; ARGUMENTS:    A => AARG
                ;               B => BARG
                ;
                ; TIMING (cycles): 4

                LOADAB MACRO    A,B

                                MOVFP   A+B0,AARG+B0   ; load lo byte of A to AARG
                                MOVFP   A+B1,AARG+B1   ; load hi byte of A to AARG
                                MOVFP   B+B0,BARG+B0   ; load lo byte of B to BARG
                                MOVFP   B+B1,BARG+B1   ; load hi byte of B to BARG

                                ENDM

                ;*****************************************************************************

                ;*****************************************************************************
                ;     ascii constants
                ;
000D            CR      set     0x0D
0018            CAN     set     0x18
0008            BS      set     0x08
0020            SP      set     0x20
000A            LF      set     0x0A
002D            MN      set     '-'
                ;
                ;*********************************************************
                ;     cmds constants and macros
                ;
                ;
0001            CHARREADY       set     0x01
                ;
                ;
0008            NUMPAR          set     0x08
                ;
                ; Response characters
                ;
0021            CMD_OK          set     '!'
003F            CMD_BAD         set     '?'
                ;
                ; Exit values
                ;
                ;
0000            HEX_SP          set     0x00
0001            HEX_MN          set     0x01
0002            HEX_CR          set     0x02
0003            HEX_CAN         set     0x03
                ;
0000            DEC_SP          set     0x00
0001            DEC_MN          set     0x01
0002            DEC_CR          set     0x02
0003            DEC_CAN         set     0x03
                ;
                ;
                ; Command characters
                ;
000D            DO_NULL                 set     CR
004D            DO_MOVE                 set     'M'     ; M
004F            DO_MODE                 set     'O'     ; O
0053            DO_SETPARAMETER         set     'S'     ; S
0052            DO_READPARAMETER        set     'R'     ; R
0043            DO_SHUTTER              set     'C'     ; C
0050            DO_READCOMPOSITION      set     'P'     ; P
0056            DO_READCOMVELOCITY      set     'V'     ; V
```


© 1993 Microchip Technology Inc.

```
0070                DO_READACTPOSITION      set     'p'     ; p
0076                DO_READACTVELOCITY      set     'v'     ; v
0058                DO_EXTERNALSTATUS       set     'X'     ; X
0059                DO_MOVESTATUS           set     'Y'     ; Y
0049                DO_READINDPOSITION      set     'I'     ; I
0048                DO_SETPOSITION          set     'H'     ; H
005A                DO_RESET                set     'Z'     ; Z
0073                DO_STOP                 set     's'     ; s
0063                DO_CAPTURE              set     'c'     ; c

                    ;*************************************************************************
                    ; NAME:         CMD_DEF
                    ;
                    ; DESCRIPTION:  Creates all the definitions for a command table data struc-
                    ;               ture. The first word is at the command character used, and
                    ;               the second word is a pointer to the function that handles
                    ;               this command function.
                    ;
                    ; ENTRY CONDITIONS:     Must be contiguous with the other entries for the
                    ;                       function to work.
                    ;
                    ; ARGUMENTS:            FUNC    command execution function
                    ;                       ROOT    NAME ROOT
                    ;

                    CMD_DEF MACRO   FUNC,ROOT

                                            DATA    ROOT
                                            DATA    FUNC
                                            ENDM

0002                CMD_ENTRY_LENGTH        set     2

                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:         CMD_START
                    ;
                    ; DESCRIPTION:  Labels the start of the command table.
                    ;
                    CMD_START MACRO LABEL

                    LABEL
                                            ENDM                    ;

                    ;*************************************************************************
                    ;*************************************************************************
                    ; NAME:         CMD_END
                    ;
                    ; DESCRIPTION:  Marks the end of the command table with an entry of 0x00
                    ;

                    CMD_END MACRO
                    ;                                                       ;
                                            DATA    0x00
                                            ENDM                            ;

                    ;*************************************************************************




                    ;
                    ;
                    ;*************************************************************************
                    ;                               PID Constantnts
                    ; define PIV parameters, computation based on errors only. Does not involve
```

© 1993 Microchip Technology Inc.

```
                        ;
                        ; No Load @ 2Khz :            Kp=3600, Ki=112, Kd= 28800, Shiftcount = 3
                        ; With Indicator Load @ 2Khz, Kp=2300 Ki= 41, Kd= 32200, Shiftcount = 4
                        ;     " " @ 4Khz              Kp=1024, Ki=8, Kd=31405 , ShiftCount = 5
                        ;     " " @ 0.5 Khz           Kp=5400, Ki=310, Kd=23400 , ShiftCount = 2
                        ;
                        ; No Load @ 1Khz             Kp=3600, Ki=192, Kd=16800, ShiftNum = 3
                        ; No Load @ 1Khz             Kp=1800, Ki=52, Kd=15600, ShiftNum = 4
                        ;
                        ; Adjust Shiftcount by maximizing Kd (for a 16 bit signed num)
                        ;
                        ;*************************************************************************

005D                    #define _KP     1800                                    ; 16 bit Kp
005E                    #define _KI     52                                      ; 16 bit Ki
005F                    #define _KV     15600                                   ; 16 bit Kv

3200                    _VEL_LIMIT      set     ((_RATED_SPEED*_ENCODER_RATE)/; 1/4 Rated sp
2000                    _ACCL_LIMIT     set     0x2000                          ; use smaller


                        if _SERVO_PID == TRUE
0004                            _SHIFTNUM       equ     4
                         endif

                        ;*************************************************************************


                                ORG     0x0                                     ; reset vector
0000 C021                       goto    Startup                                 ; startup

                                ORG     0x20
0020 C07D                       goto    InterruptPoll                           ; interrupt


                        ;*************************************************************************
                        ; NAME:         Startup
                        ;
                        ; DESCRIPTION:  This routine is called on the hardware reset or when the
                        ;               program wishes to restore initial conditions. Initiali-
                        ;               zation of run-time constants takes place here.
                        ;
                        ; RETURNS:      restart to safe and initial state
                        ;
                        ; STACK UTILIZATION: none
                        ; TIMING (in cycles): X

                        Startup

0021 8406                       bsf     _glintd                                 ; disable all
                                                AUTONO
                         ; no auto

0022 8404                       BSF     _fs0
0023 8504                       BSF     _fs1
0024 8604                       BSF     _fs2
0025 8704                       BSF     _fs3


0026 B018                       movlw   0x18                                    ; clear all
0027 4A01                       movpf   wreg,fsr0

                        memloop
0028 2900                       clrf    indf0
0029 1F01                       incfsz  fsr0
002A C028                       goto    memloop

002B 15C2                       incf    ONE
```


© 1993 Microchip Technology Inc.

```
002C B803                   movlb   bank3           ; bank3 ini
002D B03F                   movlw   TCON2_INIT
002E 770A                   movfp   wreg,tcon2

002F B07F                   movlw   PW1DCH_INIT     ; set duty
0030 720A                   movfp   wreg,pw1dch
0031 730A                   movfp   wreg,pw2dch

0032 B0C0                   movlw   PW1DCL_INIT
0033 700A                   movfp   wreg,pw1dcl
0034 710A                   movfp   wreg,pw2dcl

0035 B006                   movlw   TCON1_INIT      ; set organization of timers
0036 760A                   movfp   wreg,tcon1

0037 B802                   movlb   bank2           ; bank2 initialization

0038 B0FF                   movlw   PR1_INIT
0039 740A                   movfp   wreg,pr1        ; initialize timer1 period

003A B00F                   movlw   PR2_INIT
003B 750A                   movfp   wreg,pr2        ; initialize timer2 period


003C B800                   movlb   bank0           ; bank0 initialization

003D B080                   movlw   RTCSTA_INIT
003E 650A                   movfp   wreg,rtcsta     ; sets RTC for external input

003F B0F8                   movlw   0xF8            ; RA2 connected to BREAK Input of LMD18200
0040 0110                   movwf   porta           ; On Reset, thus pulled high breaking the motor

                    if _SERIAL_IO

0041 B090                   movlw   RCSTA_INIT
0042 730A                   movfp   wreg,rcsta      ; set receive status

0043 B020                   movlw   TXSTA_INIT      ; set transmit status
0044 750A                   movfp   wreg,txsta

0045 B019                   movlw   SPBRG_INIT      ; set baud rate
0046 770A                   movfp   wreg,spbrg
                    endif

0047 B0F3                   movlw   DDRB_INIT
0048 710A                   movfp   wreg,ddrb       ; set port B for whatever

0049 B801                   movlb   bank1           ; bank1 initialization

                    if (_SERVO_PID == TRUE)
                                MOVK16  _KP,KP

004A B008                   MOVLW   (1800) & 0xff
004B 0126                   MOVWF   KP+B0
004C B007                   MOVLW   ((1800) >> 8)
004D 0127                   MOVWF   KP+B1


                                MOVK16  _KI,KI

004E B034                   MOVLW   (52) & 0xff
004F 012A                   MOVWF   KI+B0
0050 B000                   MOVLW   ((52) >> 8)
0051 012B                   MOVWF   KI+B1


                                MOVK16  _KV,KV
```

© 1993 Microchip Technology Inc.

```
0052 B0F0                   MOVLW   (15600) & 0xff
0053 0128                   MOVWF   KV+B0
0054 B03C                   MOVLW   ((15600) >> 8)
0055 0129                   MOVWF   KV+B1


                    endif

                            MOVK16  _ACCL_LIMIT,AL

0056 B000                   MOVLW   (_ACCL_LIMIT) & 0xff
0057 0123                   MOVWF   AL+B0
0058 B020                   MOVLW   ((_ACCL_LIMIT) >> 8)
0059 0124                   MOVWF   AL+B1

                            MOVK16  _VEL_LIMIT,VL

005A B000                   MOVLW   (_VEL_LIMIT) & 0xff
005B 0120                   MOVWF   VL+B0
005C B032                   MOVLW   ((_VEL_LIMIT) >> 8)
005D 0121                   MOVWF   VL+B1



005E B004                   movlw   _SHIFTNUM
005F 01B9                   movwf   SHIFTNUM

0060 5289                   movpf   pw1dch,YPWM+B1
0061 B080                   movlw   PWMAXL          ; initialize pwm limits
0062 4A8E                   movpf   wreg,YPWMAX+B0
0063 B0FE                   movlw   PWMAXH
0064 4A8F                   movpf   wreg,YPWMAX+B1
0065 B040                   movlw   PWMINL
0066 4A8C                   movpf   wreg,YPWMIN+B0
0067 B001                   movlw   PWMINH
0068 4A8D                   movpf   wreg,YPWMIN+B1

0069 2916                   clrf    pir             ; clear flags, set indiviual interrupts
006A 2907                   clrf    intsta
006B 8517                   bsf     _tm2ie
006C 8307                   bsf     _peie

006D 8C06                   bcf     _glintd         ; enable interrupts

006E B802                   movlb   bank2

                    zeroctrs
006F 290B                   clrf    rtccl           ; clear up counter
0070 290C                   clrf    rtcch

0071 2912                   clrf    tmr31           ; clear down counter
0072 2913                   clrf    tmr3h

0073 B0FF                   movlw   0xFF
0074 170A           delay   decfsz  wreg
0075 C074                   goto    delay

0076 6A0B                   movfp   rtccl,wreg
0077 080C                   iorwf   rtcch,W
0078 0812                   iorwf   tmr31,W
0079 0813                   iorwf   tmr3h,W
007A 330A                   tstfsz  wreg
007B C06F                   goto    zeroctrs        ; motor still moving

007C C124                   goto    PollingLoop

                    ;*****************************************************************************
```


© 1993 Microchip Technology Inc.

```
                        ;*************************************************************************
                        ; NAME:       InterruptPoll
                        ;
                        InterruptPoll

007D 0138                       movwf   ISRWREG                 ; save W Reg
007E 6A04                       movfp   alusta,wreg
007F 01C7                       movwf   alustaTemp              ; save alusta
0080 4F37                       movpf   bsr,ISRBSR              ; save BSR,wreg
0081 4DC3                       movpf   tblptrl,tblptrlTemp     ; save Table Pointers
0082 4EC4                       movpf   tblptrh,tblptrhTemp
0083 A0C5                       tlrd    0,TblLatLo
0084 A2C6                       tlrd    1,TblLatHi              ; save Table Latch

0085 B801                       movlb   bank1
0086 E0E5                       call    doMPosMVel              ; calculate measured position and
                                                                ; velocity

0087 E111                       call    doExtstat               ; evaluate external status

0088 2293                       rlncf   MOVSTAT,W               ; if MOVFLAG=0 and MOVSTAT,bit7=1
0089 B501                       andlw   0x01                    ; then do premove. This is only
008A 0494                       subwf   MOVFLAG,W               ; executed once at the beginning of
008B 9F0A                       btfsc   wreg,MSB                ; each move
008C E23D                       call    doPreMove

008D 9E93                       btfsc   MOVSTAT,bit6            ; is motion continuing?
008E E30F                       call    doMove                  ; if so, do move

008F E09E                       call    doError                 ; calculate position and velocity
                                                                                ; error
0090 3390                       tstfsz  SERVOFLAG               ; test servoflag, if 0 then no servo
0091 E4AC                       call    doServo                 ; do servo

                        if      _PICMASTER_DEBUG
0092 33BB                       tstfsz  CAPFLAG
0093 E79A                       call    doCaptureRegs           ; for PIC-MASTER Trace Capture, demo
                        endif

0094 B801                       movlb   bank1

0095 2916                       clrf    pir                     ; clear all interrupt request flags

0096 A4C5                       tlwt    0,TblLatLo
0097 A6C6                       tlwt    1,TblLatHi              ; restored table latch
0098 6DC3                       movfp   tblptrlTemp,tblptrl
0099 6EC4                       movfp   tblptrhTemp,tblptrh     ; restored table pointers
009A 6F37                       movfp   ISRBSR,bsr              ; restore BSR,wreg,alusta & Table
009B 64C7                       movfp   alustaTemp,alusta
009C 6A38                       movfp   ISRWREG,wreg

009D 0005                       retfie

                        ;*************************************************************************
                        ;*************************************************************************
                        ; NAME:       doError
                        ;
                        ; DESCRIPTION: Calculates the position and velocity error.
                        ;

                        doError

                         MOV24   POSITION,POSERROR              ; calculate position error
```

```
009E 6A55                   MOVFP   POSITION+B0,wreg        ; get byte of POSITION into w
009F 4A79                   MOVPF   wreg,POSERROR+B0        ; move to POSERROR(B0)
00A0 6A56                   MOVFP   POSITION+B1,wreg        ; get byte of POSITION into w
00A1 4A7A                   MOVPF   wreg,POSERROR+B1        ; move to POSERROR(B1)
00A2 6A57                   MOVFP   POSITION+B2,wreg        ; get byte of POSITION into w
00A3 4A7B                   MOVPF   wreg,POSERROR+B2        ; move to POSERROR(B2)

                            SUB24   MPOSITION,POSERROR

00A4 6A72                   MOVFP   MPOSITION+B0,wreg       ; get lowest byte of MPOSITION into
00A5 0579                   SUBWF   POSERROR+B0             ; sub lowest byte of POSERROR,
00A6 6A73                   MOVFP   MPOSITION+B1,wreg       ; get 2nd byte of MPOSITION into
00A7 037A                   SUBWFB  POSERROR+B1             ; sub 2nd byte of POSERROR, save
00A8 6A74                   MOVFP   MPOSITION+B2,wreg       ; get 3rd byte of MPOSITION into
00A9 037B                   SUBWFB  POSERROR+B2             ; sub 3rd byte of POSERROR, save

00AA 9F7B                   btfsc   POSERROR+B2,MSB         ; saturate error to lowest 16 bits
00AB C0B7                   goto    pneg
                    ppos
00AC 6A7A                   movfp   POSERROR+B1,wreg
00AD B580                   andlw   0x80
00AE 097B                   iorwf   POSERROR+B2
00AF 290A                   clrf    wreg
00B0 327B                   cpfsgt  POSERROR+B2
00B1 C0C1                   goto    psatok
00B2 297B                   clrf    POSERROR+B2             ; clear high byte for debug purposes
00B3 B07F                   movlw   0x7F
00B4 4A7A                   movpf   wreg,POSERROR+B1
00B5 2B79                   setf    POSERROR
00B6 C0C1                   goto    psatok
                    pneg
00B7 6A7A                   movfp   POSERROR+B1,wreg
00B8 B37F                   iorlw   0x7F
00B9 0B7B                   andwf   POSERROR+B2
00BA 2B0A                   setf    wreg
00BB 307B                   cpfslt  POSERROR+B2
00BC C0C1                   goto    psatok
00BD 2B7B                   setf    POSERROR+B2             ; set high byte to 0xFF for debug
00BE 297A                   clrf    POSERROR+B1
00BF 877A                   bsf     POSERROR+B1,MSB
00C0 2979                   clrf    POSERROR
                    psatok

                            MOV24   VELOCITY,VELERROR       ; calculate velocity error

00C1 6A58                   MOVFP   VELOCITY+B0,wreg        ; get byte of VELOCITY into w
00C2 4A7C                   MOVPF   wreg,VELERROR+B0        ; move to VELERROR(B0)
00C3 6A59                   MOVFP   VELOCITY+B1,wreg        ; get byte of VELOCITY into w
00C4 4A7D                   MOVPF   wreg,VELERROR+B1        ; move to VELERROR(B1)
00C5 6A5A                   MOVFP   VELOCITY+B2,wreg        ; get byte of VELOCITY into w
00C6 4A7E                   MOVPF   wreg,VELERROR+B2        ; move to VELERROR(B2)

                            SUB24   MVELOCITY,VELERROR

00C7 6A75                   MOVFP   MVELOCITY+B0,wreg       ; get lowest byte of MVELOCITY into w
00C8 057C                   SUBWF   VELERROR+B0             ; sub lowest byte of VELERROR, save
00C9 6A76                   MOVFP   MVELOCITY+B1,wreg       ; get 2nd byte of MVELOCITY into w
00CA 037D                   SUBWFB  VELERROR+B1             ; sub 2nd byte of VELERROR, save in
00CB 6A77                   MOVFP   MVELOCITY+B2,wreg       ; get 3rd byte of MVELOCITY into w
00CC 037E                   SUBWFB  VELERROR+B2             ; sub 3rd byte of VELERROR, save in

00CD 9F7E                   btfsc   VELERROR+B2,MSB         ; saturate error to lowest 16 bits
00CE C0DA                   goto    vneg
                    vpos
```


© 1993 Microchip Technology Inc.

```
00CF 6A7D                   movfp   VELERROR+B1,wreg
00D0 B580                   andlw   0x80
00D1 097E                   iorwf   VELERROR+B2
00D2 290A                   clrf    wreg
00D3 327E                   cpfsgt  VELERROR+B2
00D4 C0E4                   goto    vsatok
00D5 297E                   clrf    VELERROR+B2
00D6 B07F                   movlw   0x7F
00D7 4A7D                   movpf   wreg,VELERROR+B1
00D8 2B7C                   setf    VELERROR
00D9 C0E4                   goto    vsatok
                    vneg
00DA 6A7D                   movfp   VELERROR+B1,wreg
00DB B37F                   iorlw   0x7F
00DC 0B7E                   andwf   VELERROR+B2
00DD 2B0A                   setf    wreg
00DE 307E                   cpfslt  VELERROR+B2
00DF C0E4                   goto    vsatok
00E0 2B7E                   setf    VELERROR+B2
00E1 297D                   clrf    VELERROR+B1
00E2 877D                   bsf     VELERROR+B1,MSB
00E3 297C                   clrf    VELERROR
                    vsatok
00E4 0002                   return
                    ;*****************************************************************************
                    ;*****************************************************************************
                    ; NAME:         doMPosMVel
                    ;
                    ; DESCRIPTION:  Calculates current position from UpCount and DownCount
                    ;
                    doMPosMVel
                    ; Do UpCounter first

                                  MOVFP16 UPCOUNT,TMP+B0              ; save old upcount

00E5 78AC                   MOVFP   UPCOUNT+B0,TMP+B0+B0    ; move UPCOUNT(B0) to TMP+B0(B0)
00E6 79AD                   MOVFP   UPCOUNT+B1,TMP+B0+B1    ; move UPCOUNT(B1) to TMP+B0(B1)


                    readUp
00E7 4C0A                   movpf   rtcch,wreg
00E8 4BAC                   movpf   rtccl,UPCOUNT+B0
00E9 310C                   cpfseq  rtcch                   ; Skip next if HI hasn't changed
00EA C0E7                   goto    readUp                  ; HI changed, re-read LO
00EB 4AAD                   movpf   wreg,UPCOUNT+B1         ; OK to store HI now

00EC 2975                   clrf    MVELOCITY+B0            ; clear bits below binary point

                            MOV16   UPCOUNT,MVELOCITY+B1    ; compute upcount increment

00ED 6AAC                   MOVFP   UPCOUNT+B0,wreg         ; get byte of UPCOUNT into w
00EE 0176                   MOVWF   MVELOCITY+B1+B0         ; move to MVELOCITY+B1(B0)
00EF 6AAD                   MOVFP   UPCOUNT+B1,wreg         ; get byte of UPCOUNT into w
00F0 0177                   MOVWF   MVELOCITY+B1+B1         ; move to MVELOCITY+B1(B1)


                            SUB16   TMP+B0,MVELOCITY+B1

00F1 6A18                   MOVFP   TMP+B0+B0,wreg          ; get lowest byte of TMP+B0 into
00F2 0576                   SUBWF   MVELOCITY+B1+B0         ; sub lowest byte of
00F3 6A19                   MOVFP   TMP+B0+B1,wreg          ; get 2nd byte of TMP+B0 into w
00F4 0377                   SUBWFB  MVELOCITY+B1+B1         ; sub 2nd byte of MVELOCITY+B1,
```


© 1993 Microchip Technology Inc.

```
                        ; Now do DownCounter

                                MOVFP16  DOWNCOUNT,TMP+B0       ; save old downcount

00F5 78AE                       MOVFP   DOWNCOUNT+B0,TMP+B0+B0  ; move DOWNCOUNT(B0) to
00F6 79AF                       MOVFP   DOWNCOUNT+B1,TMP+B0+B1  ; move DOWNCOUNT(B1) to

                        readDown
00F7 B802                       movlb   bank2   ; timers in Bank 2
00F8 530A                       movpf   tmr3h,wreg
00F9 52AE                       movpf   tmr3l,DOWNCOUNT+B0
00FA 3113                       cpfseq  tmr3h                   ; Skip next if HI hasn't changed
00FB C0F7                       goto    readDown                ; HI changed, re-read LO
00FC 4AAF                       movpf   wreg,DOWNCOUNT+B1       ; OK to store HI now

                                MOVFP16  DOWNCOUNT+B0,TMP+B2    ; compute downcount increment

00FD 7AAE                       MOVFP   DOWNCOUNT+B0+B0,TMP+B2+B0  ; move DOWNCOUNT+B0(B0) to
00FE 7BAF                       MOVFP   DOWNCOUNT+B0+B1,TMP+B2+B1  ; move DOWNCOUNT+B0(B1) to

                                SUB16   TMP+B0,TMP+B2

00FF 6A18                       MOVFP   TMP+B0+B0,wreg          ; get lowest byte of TMP+B0 into
0100 051A                       SUBWF   TMP+B2+B0               ; sub lowest byte of TMP+B2, save
0101 6A19                       MOVFP   TMP+B0+B1,wreg          ; get 2nd byte of TMP+B0 into w
0102 031B                       SUBWFB  TMP+B2+B1               ; sub 2nd byte of TMP+B2, save in

                                SUB16   TMP+B2,MVELOCITY+B1     ; compute new measured velocity

0103 6A1A                       MOVFP   TMP+B2+B0,wreg          ; get lowest byte of TMP+B2 into
0104 0576                       SUBWF   MVELOCITY+B1+B0         ; sub lowest byte of
0105 6A1B                       MOVFP   TMP+B2+B1,wreg          ; get 2nd byte of TMP+B2 into w
0106 0377                       SUBWFB  MVELOCITY+B1+B1         ; sub 2nd byte of MVELOCITY+B1,

0107 2978                       clrf    MVELOCITY+B3            ; sign extend measured velocity
0108 9F77                       btfsc   MVELOCITY+B2,MSB        ; 24 bit addition to measured
0109 2B78                       setf    MVELOCITY+B3

                                ADD24   MVELOCITY+B1,MPOSITION  ; compute new measured position

010A 6A76                       MOVFP   MVELOCITY+B1+B0,wreg    ; get lowest byte of MVELOCITY+B1
010B 0F72                       ADDWF   MPOSITION+B0            ; add lowest byte of MPOSITION,
010C 6A77                       MOVFP   MVELOCITY+B1+B1,wreg    ; get 2nd byte of MVELOCITY+B1
010D 1173                       ADDWFC  MPOSITION+B1            ; add 2nd byte of MPOSITION, save
010E 6A78                       MOVFP   MVELOCITY+B1+B2,wreg    ; get 3rd byte of MVELOCITY+B1
010F 1174                       ADDWFC  MPOSITION+B2            ; add 3rd byte of MPOSITION, save

                                                                ; delta position = measured
0110 0002                       return

                        ;*****************************************************************
                        ; NAME:         doExtstat
                        ;
                        ; DESCRIPTION:  Get +limit,-limit,GPI from PORTB and set in EXTSTAT
                        ;
                        doExtstat
0111 9407                       btfss   _intir
0112 C11B                       goto    otherbits
                                MOV24   MPOSITION,INDEXPOS
```


© 1993 Microchip Technology Inc.

```
0113 6A72                   MOVFP   MPOSITION+B0,wreg          ; get byte of MPOSITION into w
0114 4AB6                   MOVPF   wreg,INDEXPOS+B0          ; move to INDEXPOS(B0)
0115 6A73                   MOVFP   MPOSITION+B1,wreg          ; get byte of MPOSITION into w
0116 4AB7                   MOVPF   wreg,INDEXPOS+B1          ; move to INDEXPOS(B1)
0117 6A74                   MOVFP   MPOSITION+B2,wreg          ; get byte of MPOSITION into w
0118 4AB8                   MOVPF   wreg,INDEXPOS+B2          ; move to INDEXPOS(B2)


0119 8C07                   bcf     _intir
011A 8792                   bsf     EXTSTAT,MSB

                    otherbits
011B B800                   movlb   bank0                     ; get +limit,-limit and GPI
011C 6A12                   movfp   portb,wreg
011D 190A                   rrcf    wreg                      ; arrange in correct bit posi
011E B561                   andlw   0x61
011F 4A18                   movpf   wreg,TMP
0120 1D18                   swapf   TMP
0121 0818                   iorwf   TMP,W
0122 0992                   iorwf   EXTSTAT                   ; set in EXTSTAT

0123 0002                   return

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME:                 PollingLoop
                    ;
                    ; DESCRIPTION:          The actual polling loop called after the board's
                    ;                       initialization
                    ;
                    ; ENTRY CONDITIONS:     System globals and hardware initialized and the
                    ;                       interrupt processes started.
                    ;
                    PollingLoop

                    if _SERIAL_IO
0124 E557                   call    IdleFunction
0125 E681                   call    GetChk
0126 31C2                   cpfseq  ONE                       ; GetChk, is receive buffer full?
0127 C124                   goto    PollingLoop

0128 E676                   call    GetChar                   ; if so, get character
0129 4A39                   movpf   wreg,CMDCHAR              ; put in CMDCHAR
012A C559                   goto    DoCommand
                     else
                            clrwdt
                            goto    PollingLoop               ; wait for Interrupt
                     endif

                    ;*****************************************************************************

                                        include "mult.asm"    ; Double Precision
                    Multiplication Routine
                    ;*****************************************************************************
                    ;                       Double Precision Multiplication Routine
                    ;
                    ; NAME:         Dmult
                    ;
                    ; DESCRIPTION:  Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
                    ;
                    ;       (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
                    ;       (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
                    ;       (c) call Dmult
                    ;       (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
                    ;
                    ;       In the signed case, a savings of 9 clks can be realized by choosing
```

© 1993 Microchip Technology Inc.

```
                ;       BARG as the positive factor in the product when possible.
                ;
                ; TIMING (worst case):  unsigned:                  173 clks
                ;                       signed:    if BARG positive:  170 clks
                ;                                  if BARG negative:  179 clks
                ;
                ;*****************************************************************************

0001            SIGNED  equ     TRUE            ; Set This To 'TRUE' for signed multiply
                                                ; and 'FALSE' for unsigned.
                ;*****************************************************************************
                ;               Multiplication Macro
                ;*****************************************************************************
                ;
                ; TIMING:       unsigned:     11+7*10+8*11    = 169 clks
                ;(worst case)   signed:       11+7*10+7*11+5 = 163 clks
                ;
                MULTMAC  MACRO
                        variable i
                                i = 0
                          if   SIGNED
                             while i < 15
                          else
                             while i < 16
                          endif
                             if i < 8
                               btfsc    BARG+B0,i          ; test low byte
                               else
                               btfsc    BARG+B1,i-8        ; test high byte
                         endif
                                goto    add#v(i)
                           if i < 8
                               rlcf     DPX+B3,W           ; rotate sign into carry bit
                               rrcf     DPX+B3             ; for i < 8, no meaningful
                               rrcf     DPX+B2             ; are in DPX+B0
                               rrcf     DPX+B1
                             else
                               rlcf     DPX+B3,W           ; rotate sign into carry bit
                               rrcf     DPX+B3
                               rrcf     DPX+B2
                               rrcf     DPX+B1
                               rrcf     DPX+B0
                             endif
                               i = i+1
                             endw
                               clrf     DPX+B0             ; if we get here, BARG = 0
                               return
                        add0
                               movfp    AARG+B0,wreg
                               addwf    DPX+B2             ;add lsb
                               movfp    AARG+B1,wreg
                               addwfc   DPX+B3             ;add msb
                               rlcf     AARG+B1,W          ; rotate sign into carry bit
                               rrcf     DPX+B3             ; for i < 8, no meaningful
                               rrcf     DPX+B2             ; are in DPX+B0
                               rrcf     DPX+B1
                          i = 1
                          if   SIGNED
                               while i < 15
                          else
                             while i < 16
                          endif
                             if i < 8
                               btfss    BARG+B0,i          ; test low byte
                             else
                               btfss    BARG+B1,i-8        ; test high byte
                             endif
                               goto     noadd#v(i)
```


© 1993 Microchip Technology Inc.

```
                                add#v(i)
                                movfp   AARG+B0,wreg
                                addwf   DPX+B2              ;add lsb
                                movfp   AARG+B1,wreg
                                addwfc  DPX+B3              ;add msb
                            noadd#v(i)
                                if i < 8
                                  rlcf    AARG+B1,W         ; rotate sign into carry bit
                                  rrcf    DPX+B3            ; for i < 8, no meaningful
                                  rrcf    DPX+B2            ; are in DPX+B0
                                  rrcf    DPX+B1
                                else
                                  rlcf    AARG+B1,W         ; rotate sign into carry bit
                                  rrcf    DPX+B3
                                  rrcf    DPX+B2
                                  rrcf    DPX+B1
                                  rrcf    DPX+B0
                                endf
                                i = i+1
                                endw
                             if   SIGNED
                                  rlcf    AARG+B1,W         ; since BARG is always made
                                  rrcf    DPX+B3            ; the last bit is known to be
                                  rrcf    DPX+B2
                                  rrcf    DPX+B1
                                  rrcf    DPX+B0
                              endif
                            ENDM
                                                            ; Double Precision Multiply (
                                                            ; ( AARG*BARG -> : 32 bit
                    ;
                    Dmult
                         if   SIGNED
012B 971F                btfss   BARG+B1,MSB                ; test sign of BARG
012C C137                goto    argsok                     ; if positive, ok
                                   NEG16   AARG+B0         ; if negative, then negate

012D 131C                COMF    AARG+B0+B0
012E 131D                COMF    AARG+B0+B1
012F 290A                CLRF    wreg
0130 151C                INCF    AARG+B0+B0
0131 111D                ADDWFC  AARG+B0+B1
                                   NEG16   BARG+B0         ; AARG and BARG
0132 131E                COMF    BARG+B0+B0
0133 131F                COMF    BARG+B0+B1
0134 290A                CLRF    wreg
0135 151E                INCF    BARG+B0+B0
0136 111F                ADDWFC  BARG+B0+B1
                    endif
                    argsok
0137 291B                clrf    DPX+B3          ; clear initial partial product
0138 291A                clrf    DPX+B2
                 MULTMAC                         ; use macro for multiplication
0000                     variable i
0000                          i = 0
                           if   SIGNED
                              while i < 15
                                                  else
                              while i < 16
                           endif
                              if i < 8
                                btfsc     BARG+B0,i       ; test low byte
                                  else
                                btfsc     BARG+B1,i-8     ; test high byte
                              endif
                                goto    add#v(i)
                              if i < 8
                                rlcf    DPX+B3,W          ; rotate sign into carry bit
                                rrcf    DPX+B3            ; for i < 8, no meaningful bits
```

© 1993 Microchip Technology Inc.

```
                                  rrcf    DPX+B2          ; are in DPX+B0
                                  rrcf    DPX+B1
                               else
                                  rlcf    DPX+B3,W        ; rotate sign into carry bit
                                  rrcf    DPX+B3
                                  rrcf    DPX+B2
                                  rrcf    DPX+B1
                                  rrcf    DPX+B0
                                  endif
                        i = i+1
                               endw
                              if i < 8
0139 981E                        btfsc    BARG+B0,i       ; test low byte
                                   else
                                 btfsc     BARG+B1,i-8     ; test high byte
                              endif
013A C19C                        goto    add0
                              if i < 8
013B 1A1B                        rlcf    DPX+B3,W        ; rotate sign into carry bit
013C 191B                        rrcf    DPX+B3          ; for i < 8, no meaningful
013D 191A                        rrcf    DPX+B2          ; are in DPX+B0
013E 1919                        rrcf    DPX+B1
                              else
                                 rlcf    DPX+B3,W        ; rotate sign into carry bit
                                 rrcf    DPX+B3
                                 rrcf    DPX+B2
                                 rrcf    DPX+B1
                                 rrcf    DPX+B0
                              endif
0001                             i = i+1
                              if i < 8
013F 991E                        btfsc    BARG+B0,i       ; test low byte
                                   else
                                 btfsc     BARG+B1,i-8     ; test high byte
                              endif
0140 C1A6                        goto    add1
                              if i < 8
0141 1A1B                        rlcf    DPX+B3,W        ; rotate sign into carry bit
0142 191B                        rrcf    DPX+B3          ; for i < 8, no meaningful
0143 191A                        rrcf    DPX+B2          ; are in DPX+B0
0144 1919                        rrcf    DPX+B1
                              else
                                 rlcf    DPX+B3,W        ; rotate sign into carry bit
                                 rrcf    DPX+B3
                                 rrcf    DPX+B2
                                 rrcf    DPX+B1
                                 rrcf    DPX+B0

                              endif

0002                             i = i+1
                              if i < 8
0145 9A1E                        btfsc    BARG+B0,i       ; test low byte
                                   else
                                 btfsc     BARG+B1,i-8     ; test high byte

                              endif

0146 C1B0                        goto    add2
                              if i < 8
0147 1A1B                        rlcf    DPX+B3,W        ; rotate sign into carry bit
0148 191B                        rrcf    DPX+B3          ; for i < 8, no meaningful
0149 191A                        rrcf    DPX+B2          ; are in DPX+B0
014A 1919                        rrcf    DPX+B1
                              else
                                 rlcf    DPX+B3,W        ; rotate sign into carry bit
                                 rrcf    DPX+B3
                                 rrcf    DPX+B2
                                 rrcf    DPX+B1
```


© 1993 Microchip Technology Inc.

```
                              rrcf    DPX+B0
                            endif
0003                           i = i+1
                            if i < 8
014B 9B1E                      btfsc     BARG+B0,i          ; test low byte
                                 else
                               btfsc     BARG+B1,i-8        ; test high byte
                            endif
014C C1BA                      goto    add3
                            if i < 8
014D 1A1B                      rlcf    DPX+B3,W           ; rotate sign into carry bit
014E 191B                      rrcf    DPX+B3             ; for i < 8, no meaningful
014F 191A                      rrcf    DPX+B2             ; are in DPX+B0
0150 1919                      rrcf    DPX+B1
                            else
                               rlcf    DPX+B3,W           ; rotate sign into carry bit
                               rrcf    DPX+B3
                               rrcf    DPX+B2
                               rrcf    DPX+B1
                               rrcf    DPX+B0
                            endif
0004                           i = i+1
                            if i < 8
0151 9C1E                      btfsc     BARG+B0,i          ; test low byte
                                 else
                               btfsc     BARG+B1,i-8        ; test high byte
                            endif
0152 C1C4                      goto    add4
                            if i < 8
0153 1A1B                      rlcf    DPX+B3,W           ; rotate sign into carry bit
0154 191B                      rrcf    DPX+B3             ; for i < 8, no meaningful
0155 191A                      rrcf    DPX+B2             ; are in DPX+B0
0156 1919                      rrcf    DPX+B1
                            else
                               rlcf    DPX+B3,W           ; rotate sign into carry bit
                               rrcf    DPX+B3
                               rrcf    DPX+B2
                               rrcf    DPX+B1
                               rrcf    DPX+B0
                            endif
0005                           i = i+1
                            if i < 8
0157 9D1E                      btfsc     BARG+B0,i          ; test low byte
                                 else
                               btfsc     BARG+B1,i-8        ; test high byte
                            endif
0158 C1CE                      goto    add5
                            if i < 8
0159 1A1B                      rlcf    DPX+B3,W           ; rotate sign into carry bit
015A 191B                      rrcf    DPX+B3             ; for i < 8, no meaningful
015B 191A                      rrcf    DPX+B2             ; are in DPX+B0
015C 1919                      rrcf    DPX+B1
                            else
                               rlcf    DPX+B3,W           ; rotate sign into carry bit
                               rrcf    DPX+B3
                               rrcf    DPX+B2
                               rrcf    DPX+B1
                               rrcf    DPX+B0
                            endif
0006                           i = i+1
                            if i < 8
015D 9E1E                      btfsc     BARG+B0,i          ; test low byte
                                 else
                               btfsc     BARG+B1,i-8        ; test high byte
                            endif
015E C1D8                      goto    add6
                            if i < 8
015F 1A1B                      rlcf    DPX+B3,W           ; rotate sign into carry bit
0160 191B                      rrcf    DPX+B3             ; for i < 8, no meaningful
```


© 1993 Microchip Technology Inc.

```
0161 191A                       rrcf    DPX+B2          ; are in DPX+B0
0162 1919                       rrcf    DPX+B1
                              else
                                rlcf    DPX+B3,W        ; rotate sign into carry bit
                                rrcf    DPX+B3
                                rrcf    DPX+B2
                                rrcf    DPX+B1
                                rrcf    DPX+B0
                              endif
0007                            i = i+1
                              if i < 8
0163 9F1E                       btfsc   BARG+B0,i       ; test low byte
                                  else
                                btfsc   BARG+B1,i-8     ; test high byte
                              endif
0164 C1E2                       goto    add7
                              if i < 8
0165 1A1B                       rlcf    DPX+B3,W        ; rotate sign into carry bit
0166 191B                       rrcf    DPX+B3          ; for i < 8, no meaningful
0167 191A                       rrcf    DPX+B2          ; are in DPX+B0
0168 1919                       rrcf    DPX+B1
                              else
                                rlcf    DPX+B3,W        ; rotate sign into carry bit
                                rrcf    DPX+B3
                                rrcf    DPX+B2
                                rrcf    DPX+B1
                                rrcf    DPX+B0
                              endif
0008                            i = i+1
                              if i < 8
                                btfsc   BARG+B0,i       ; test low byte
                                   else
0169 981F                       btfsc   BARG+B1,i-8     ; test high byte
                              endif
016A C1EC                       goto    add8
                              if i < 8
                                rlcf    DPX+B3,W        ; rotate sign into carry bit
                                rrcf    DPX+B3          ; for i < 8, no meaningful
                                rrcf    DPX+B2          ; are in DPX+B0
                                rrcf    DPX+B1
                              else
016B 1A1B                       rlcf    DPX+B3,W        ; rotate sign into carry bit
016C 191B                       rrcf    DPX+B3
016D 191A                       rrcf    DPX+B2
016E 1919                       rrcf    DPX+B1
016F 1918                       rrcf    DPX+B0
                              endif
0009                            i = i+1
                              if i < 8
                                btfsc   BARG+B0,i       ; test low byte
                                else
0170 991F                       btfsc   BARG+B1,i-8     ; test high byte
                              endif
0171 C1F7                       goto    add9
                              if i < 8
                                rlcf    DPX+B3,W        ; rotate sign into carry bit
                                rrcf    DPX+B3          ; for i < 8, no meaningful
                                rrcf    DPX+B2          ; are in DPX+B0
                                rrcf    DPX+B1
                              else
0172 1A1B                       rlcf    DPX+B3,W        ; rotate sign into carry bit
0173 191B                       rrcf    DPX+B3
0174 191A                       rrcf    DPX+B2
0175 1919                       rrcf    DPX+B1
0176 1918                       rrcf    DPX+B0
                              endif
000A                            i = i+1
                              if i < 8
                                btfsc   BARG+B0,i       ; test low byte
```


© 1993 Microchip Technology Inc.

```
                                  else
0177 9A1F                           btfsc     BARG+B1,i-8       ; test high byte
                                  endif
0178 C202                           goto    add10
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0179 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
017A 191B                           rrcf    DPX+B3
017B 191A                           rrcf    DPX+B2
017C 1919                           rrcf    DPX+B1
017D 1918                           rrcf    DPX+B0
                                  endif
000B                              i = i+1
                                  if i < 8
                                    btfsc     BARG+B0,i         ; test low byte
                                      else
017E 9B1F                           btfsc     BARG+B1,i-8       ; test high byte
                                  endif
017F C20D                           goto    add11
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0180 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
0181 191B                           rrcf    DPX+B3
0182 191A                           rrcf    DPX+B2
0183 1919                           rrcf    DPX+B1
0184 1918                           rrcf    DPX+B0
                                  endif
000C                              i = i+1
                                  if i < 8
                                    btfsc     BARG+B0,i         ; test low byte
                                      else
0185 9C1F                           btfsc     BARG+B1,i-8       ; test high byte
                                  endif
0186 C218                           goto    add12
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0187 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
0188 191B                           rrcf    DPX+B3
0189 191A                           rrcf    DPX+B2
018A 1919                           rrcf    DPX+B1
018B 1918                           rrcf    DPX+B0
                                  endif
000D                              i = i+1
                                  if i < 8
                                    btfsc     BARG+B0,i         ; test low byte
                                      else
018C 9D1F                           btfsc     BARG+B1,i-8       ; test high byte
                                  endif
018D C223                           goto    add13
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
018E 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
018F 191B                           rrcf    DPX+B3
```


© 1993 Microchip Technology Inc.

```
0190 191A                           rrcf    DPX+B2
0191 1919                           rrcf    DPX+B1
0192 1918                           rrcf    DPX+B0
                                  endif
000E                                i = i+1
                                  if i < 8
                                     btfsc      BARG+B0,i          ; test low byte
                                       else
0193 9E1F                            btfsc      BARG+B1,i-8        ; test high byte
                                  endif
0194 C22E                           goto    add14
                                  if i < 8
                                     rlcf    DPX+B3,W            ; rotate sign into carry bit
                                     rrcf    DPX+B3              ; for i < 8, no meaningful
                                     rrcf    DPX+B2              ; are in DPX+B0
                                     rrcf    DPX+B1
                                  else
0195 1A1B                            rlcf    DPX+B3,W           ; rotate sign into carry bit
0196 191B                           rrcf    DPX+B3
0197 191A                           rrcf    DPX+B2
0198 1919                           rrcf    DPX+B1
0199 1918                           rrcf    DPX+B0
                                  endif
000F                                i = i+1
019A 2918                           clrf    DPX+B0              ; if we get here, BARG = 0
019B 0002                           return
                    add0
019C 6A1C                           movfp   AARG+B0,wreg
019D 0F1A                           addwf   DPX+B2              ;add lsb
019E 6A1D                           movfp   AARG+B1,wreg
019F 111B                           addwfc  DPX+B3 ;add msb
01A0 1A1D                           rlcf    AARG+B1,W           ; rotate sign into carry bit
01A1 191B                           rrcf    DPX+B3              ; for i < 8, no meaningful
01A2 191A                           rrcf    DPX+B2              ; are in DPX+B0
01A3 1919                           rrcf    DPX+B1
0001                              i = 1
                                  if   SIGNED
                                         while i < 15
                                   else
                                     while i < 16
                                  endif
                                     if i < 8
                                       btfss      BARG+B0,i   ; test low byte
                                     else
                                       btfss      BARG+B1,i-8 ; test high byte
                                     endif
                                       goto    noadd#v(i)
                    add#v(i)
                                       movfp   AARG+B0,wreg
                                       addwf   DPX+B2 ;add lsb
                                       movfp   AARG+B1,wreg
                                       addwfc  DPX+B3 ;add msb
                    noadd#v(i)
                                     if i < 8
                                       rlcf    AARG+B1,W      ; rotate sign into carry bit
                                       rrcf    DPX+B3         ; for i < 8, no meaningful bits
                                       rrcf    DPX+B2         ; are in DPX+B0
                                       rrcf    DPX+B1
                                     else
                                       rlcf    AARG+B1,W      ; rotate sign into carry bit
                                       rrcf    DPX+B3
                                       rrcf    DPX+B2
                                       rrcf    DPX+B1
                                       rrcf    DPX+B0
                                     endif
                                     i = i+1
                                     endw
                                   if i < 8
01A4 911E                            btfss      BARG+B0,i      ; test low byte
```


© 1993 Microchip Technology Inc.

```
                                  else
                                     btfss      BARG+B1,i-8  ; test high byte
                                  endif
01A5 C1AA                            goto     noadd1
                        add1
01A6 6A1C                            movfp    AARG+B0,wreg
01A7 0F1A                            addwf    DPX+B2          ;add lsb
01A8 6A1D                            movfp    AARG+B1,wreg
01A9 111B                            addwfc   DPX+B3          ;add msb
                        noadd1
                                  if i < 8
01AA 1A1D                            rlcf     AARG+B1,W       ; rotate sign into carry bit
01AB 191B                            rrcf     DPX+B3          ; for i < 8, no meaningful bits
01AC 191A                            rrcf     DPX+B2          ; are in DPX+B0
01AD 1919                            rrcf     DPX+B1
                                  else
                                     rlcf     AARG+B1,W       ; rotate sign into carry bit
                                     rrcf     DPX+B3
                                     rrcf     DPX+B2
                                     rrcf     DPX+B1
                                     rrcf     DPX+B0
                                  endif
0002                              i = i+1
                                  if i < 8
01AE 921E                            btfss      BARG+B0,i          ; test low byte
                                  else
                                     btfss      BARG+B1,i-8        ; test high byte
                                  endif
01AF C1B4                            goto     noadd2                    add2
01B0 6A1C                            movfp    AARG+B0,wreg
01B1 0F1A                            addwf    DPX+B2             ;add lsb
01B2 6A1D                            movfp    AARG+B1,wreg
01B3 111B                            addwfc   DPX+B3             ;add msb
                        noadd2
                                  if i < 8
01B4 1A1D                            rlcf     AARG+B1,W          ; rotate sign into carry bit
01B5 191B                            rrcf     DPX+B3             ; for i < 8, no meaningful
01B6 191A                            rrcf     DPX+B2             ; are in DPX+B0
01B7 1919                            rrcf     DPX+B1
                                  else
                                     rlcf     AARG+B1,W          ; rotate sign into carry bit
                                     rrcf     DPX+B3
                                     rrcf     DPX+B2
                                     rrcf     DPX+B1
                                     rrcf     DPX+B0
                                  endif
0003                              i = i+1
                                  if i < 8
01B8 931E                            btfss      BARG+B0,i          ; test low byte
                                  else
                                     btfss      BARG+B1,i-8        ; test high byte
                                  endif
01B9 C1BE                            goto     noadd3
                        add3
01BA 6A1C                            movfp    AARG+B0,wreg
01BB 0F1A                            addwf    DPX+B2             ;add lsb
01BC 6A1D                            movfp    AARG+B1,wreg
01BD 111B                            addwfc   DPX+B3             ;add msb
                        noadd3
                                  if i < 8
01BE 1A1D                            rlcf     AARG+B1,W          ; rotate sign into carry bit
01BF 191B                            rrcf     DPX+B3             ; for i < 8, no meaningful
01C0 191A                            rrcf     DPX+B2             ; are in DPX+B0
01C1 1919                            rrcf     DPX+B1
                                  else
                                     rlcf     AARG+B1,W          ; rotate sign into carry bit
                                     rrcf     DPX+B3
                                     rrcf     DPX+B2
                                     rrcf     DPX+B1
                                     rrcf     DPX+B0
```

4

© 1993 Microchip Technology Inc.

```
                    endif
0004                i = i+1
                    if i < 8
01C2 941E              btfss    BARG+B0,i        ; test low byte
                    else
                       btfss    BARG+B1,i-8      ; test high byte
                    endif
01C3 C1C8              goto     noadd4
                 add4
01C4 6A1C              movfp    AARG+B0,wreg
01C5 0F1A              addwf    DPX+B2           ;add lsb
01C6 6A1D              movfp    AARG+B1,wreg
01C7 111B              addwfc   DPX+B3           ;add msb
                 noadd4
                    if i < 8
01C8 1A1D              rlcf     AARG+B1,W        ; rotate sign into carry bit
01C9 191B              rrcf     DPX+B3           ; for i < 8, no meaningful
01CA 191A              rrcf     DPX+B2           ; are in DPX+B0
01CB 1919              rrcf     DPX+B1
                    else
                       rlcf     AARG+B1,W        ; rotate sign into carry bit
                       rrcf     DPX+B3
                       rrcf     DPX+B2
                       rrcf     DPX+B1
                       rrcf     DPX+B0
                    endif
0005                i = i+1
                    if i < 8
01CC 951E              btfss    BARG+B0,i        ; test low byte
                    else
                       btfss    BARG+B1,i-8      ; test high byte
                    endif
01CD C1D2              goto     noadd5
                 add5
01CE 6A1C              movfp    AARG+B0,wreg
01CF 0F1A              addwf    DPX+B2  ;add lsb
01D0 6A1D              movfp    AARG+B1,wreg
01D1 111B              addwfc   DPX+B3  ;add msb
                 noadd5
                    if i < 8
01D2 1A1D              rlcf     AARG+B1,W        ; rotate sign into carry bit
01D3 191B              rrcf     DPX+B3           ; for i < 8, no meaningful
01D4 191A              rrcf     DPX+B2           ; are in DPX+B0
01D5 1919              rrcf     DPX+B1
                    else
                       rlcf     AARG+B1,W        ; rotate sign into carry bit
                       rrcf     DPX+B3
                       rrcf     DPX+B2
                       rrcf     DPX+B1

                       rrcf     DPX+B0



                    endif

0006                i = i+1

                    if i < 8
01D6 961E              btfss    BARG+B0,i        ; test low byte
                    else
                       btfss    BARG+B1,i-8      ; test high byte

                    endif

01D7 C1DC              goto     noadd6
                 add6
01D8 6A1C              movfp    AARG+B0,wreg
```


© 1993 Microchip Technology Inc.

```
01D9 0F1A                          addwf   DPX+B2  ;add lsb
01DA 6A1D                          movfp   AARG+B1,wreg
01DB 111B                          addwfc  DPX+B3  ;add msb
                       noadd6
                                 if i < 8
01DC 1A1D                          rlcf    AARG+B1,W          ; rotate sign into carry bit
01DD 191B                          rrcf    DPX+B3             ; for i < 8, no meaningful
01DE 191A                          rrcf    DPX+B2             ; are in DPX+B0
01DF 1919                          rrcf    DPX+B1
                                 else
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                 endif
0007                             i = i+1
                                 if i < 8
01E0 971E                          btfss     BARG+B0,i        ; test low byte
                                 else
                                    btfss     BARG+B1,i-8      ; test high byte
                                 endif
01E1 C1E6                          goto    noadd7
                       add7
01E2 6A1C                          movfp   AARG+B0,wreg
01E3 0F1A                          addwf   DPX+B2  ;add lsb
01E4 6A1D                          movfp   AARG+B1,wreg
01E5 111B                          addwfc  DPX+B3  ;add msb
                       noadd7
                                 if i < 8
01E6 1A1D                          rlcf    AARG+B1,W          ; rotate sign into carry bit
01E7 191B                          rrcf    DPX+B3             ; for i < 8, no meaningful
01E8 191A                          rrcf    DPX+B2             ; are in DPX+B0
01E9 1919                          rrcf    DPX+B1
                                 else
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                 endif
0008                             i = i+1
                                 if i < 8
                                    btfss     BARG+B0,i        ; test low byte
                                 else
01EA 901F                           btfss     BARG+B1,i-8      ; test high byte
                                 endif
01EB C1F0                          goto    noadd8
                       add8
01EC 6A1C                          movfp   AARG+B0,wreg
01ED 0F1A                          addwf   DPX+B2  ;add lsb
01EE 6A1D                          movfp   AARG+B1,wreg
01EF 111B                          addwfc  DPX+B3  ;add msb
                       noadd8
                                 if i < 8
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3             ; for i < 8, no meaningful
                                    rrcf    DPX+B2             ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
01F0 1A1D                           rlcf    AARG+B1,W          ; rotate sign into carry bit
01F1 191B                           rrcf    DPX+B3
01F2 191A                           rrcf    DPX+B2
01F3 1919                           rrcf    DPX+B1
01F4 1918                           rrcf    DPX+B0
                                 endif
0009                             i = i+1
```


© 1993 Microchip Technology Inc.

```
                                    if i < 8
                                       btfss    BARG+B0,i          ; test low byte
                                    else
01F5 911F                              btfss    BARG+B1,i-8        ; test high byte
                                    endif
01F6 C1FB                              goto     noadd9
                          add9
01F7 6A1C                              movfp    AARG+B0,wreg
01F8 0F1A                              addwf    DPX+B2 ;add lsb
01F9 6A1D                              movfp    AARG+B1,wreg
01FA 111B                              addwfc   DPX+B3 ;add msb
                          noadd9
                                    if i < 8
                                       rlcf     AARG+B1,W          ; rotate sign into carry bit
                                       rrcf     DPX+B3             ; for i < 8, no meaningful
                                       rrcf     DPX+B2             ; are in DPX+B0
                                       rrcf     DPX+B1
                                    else
01FB 1A1D                              rlcf     AARG+B1,W          ; rotate sign into carry bit
01FC 191B                              rrcf     DPX+B3
01FD 191A                              rrcf     DPX+B2
01FE 1919                              rrcf     DPX+B1
01FF 1918                              rrcf     DPX+B0
                                    endif
000A                                i = i+1
                                    if i < 8
                                       btfss    BARG+B0,i          ; test low byte
                                    else
0200 921F                              btfss    BARG+B1,i-8        ; test high byte
                                    endif
0201 C206                              goto     noadd10
                          add10
0202 6A1C                              movfp    AARG+B0,wreg
0203 0F1A                              addwf    DPX+B2 ;add lsb
0204 6A1D                              movfp    AARG+B1,wreg
0205 111B                              addwfc   DPX+B3 ;add msb
                          noadd10
                                    if i < 8
                                       rlcf     AARG+B1,W          ; rotate sign into carry bit
                                       rrcf     DPX+B3             ; for i < 8, no meaningful
                                       rrcf     DPX+B2             ; are in DPX+B0
                                       rrcf     DPX+B1
                                    else
0206 1A1D                              rlcf     AARG+B1,W          ; rotate sign into carry bit
0207 191B                              rrcf     DPX+B3
0208 191A                              rrcf     DPX+B2
0209 1919                              rrcf     DPX+B1
020A 1918                              rrcf     DPX+B0
                                    endif
000B                                i = i+1
                                    if i < 8
                                       btfss    BARG+B0,i          ; test low byte
                                    else
020B 931F                              btfss    BARG+B1,i-8        ; test high byte
                                    endif
020C C211                              goto     noadd11
                          add11
020D 6A1C                              movfp    AARG+B0,wreg
020E 0F1A                              addwf    DPX+B2 ;add lsb
020F 6A1D                              movfp    AARG+B1,wreg
0210 111B                              addwfc   DPX+B3 ;add msb
                          noadd11
                                    if i < 8
                                       rlcf     AARG+B1,W          ; rotate sign into carry bit
                                       rrcf     DPX+B3             ; for i < 8, no meaningful
                                       rrcf     DPX+B2             ; are in DPX+B0
                                       rrcf     DPX+B1
                                    else
```


© 1993 Microchip Technology Inc.

```
0211 1A1D                          rlcf    AARG+B1,W          ; rotate sign into carry bit
0212 191B                          rrcf    DPX+B3
0213 191A                          rrcf    DPX+B2
0214 1919                          rrcf    DPX+B1
0215 1918                          rrcf    DPX+B0
                                   endif
000C                               i = i+1
                                   if i < 8
                                      btfss    BARG+B0,i        ; test low byte
                                   else
0216 941F                             btfss    BARG+B1,i-8      ; test high byte
                                   endif
0217 C21C                          goto    noadd12
                    add12
0218 6A1C                          movfp   AARG+B0,wreg
0219 0F1A                          addwf   DPX+B2  ;add lsb
021A 6A1D                          movfp   AARG+B1,wreg
021B 111B                          addwfc  DPX+B3  ;add msb
                    noadd12
                                   if i < 8
                                      rlcf    AARG+B1,W       ; rotate sign into carry bit
                                      rrcf    DPX+B3          ; for i < 8, no meaningful
                                      rrcf    DPX+B2          ; are in DPX+B0
                                      rrcf    DPX+B1
                                   else
021C 1A1D                          rlcf    AARG+B1,W          ; rotate sign into carry bit
021D 191B                          rrcf    DPX+B3
021E 191A                          rrcf    DPX+B2
021F 1919                          rrcf    DPX+B1
0220 1918                          rrcf    DPX+B0
                                   endif
000D                               i = i+1
                                   if i < 8
                                      btfss    BARG+B0,i       ; test low byte
                                   else
221 951F                              btfss    BARG+B1,i-8     ; test high byte
                                   endif
0222 C227                          goto    noadd13
                    add13
0223 6A1C                          movfp   AARG+B0,wreg
0224 0F1A                          addwf   DPX+B2  ;add lsb
0225 6A1D                          movfp   AARG+B1,wreg
0226 111B                          addwfc  DPX+B3  ;add msb
                    noadd13
                                   if i < 8
                                      rlcf    AARG+B1,W       ; rotate sign into carry bit
                                      rrcf    DPX+B3          ; for i < 8, no meaningful
                                      rrcf    DPX+B2          ; are in DPX+B0
                                      rrcf    DPX+B1
                                   else
0227 1A1D                          rlcf    AARG+B1,W          ; rotate sign into carry bit
0228 191B                          rrcf    DPX+B3
0229 191A                          rrcf    DPX+B2
022A 1919                          rrcf    DPX+B1
022B 1918                          rrcf    DPX+B0
                                   endif
000E                               i = i+1
                                   if i < 8
                                      btfss    BARG+B0,i       ; test low byte
                                   else
022C 961F                             btfss    BARG+B1,i-8     ; test high byte
                                   endif
022D C232                          goto    noadd14
                    add14
022E 6A1C                          movfp   AARG+B0,wreg
022F 0F1A                          addwf   DPX+B2             ;add lsb
0230 6A1D                          movfp   AARG+B1,wreg
0231 111B                          addwfc  DPX+B3             ;add msb
```


© 1993 Microchip Technology Inc.

```
                        noadd14
                                    if i < 8
                                       rlcf    AARG+B1,W          ; rotate sign into carry bit
                                       rrcf    DPX+B3             ; for i < 8, no meaningful
                                       rrcf    DPX+B2             ; are in DPX+B0
                                       rrcf    DPX+B1
                                    else
0232 1A1D                              rlcf    AARG+B1,W          ; rotate sign into carry bit
0233 191B                              rrcf    DPX+B3
0234 191A                              rrcf    DPX+B2
0235 1919                              rrcf    DPX+B1
0236 1918                              rrcf    DPX+B0
                                    endif
000F                                i = i+1
                                 if   SIGNED
0237 1A1D                              rlcf    AARG+B1,W          ; since BARG is always made
0238 191B                              rrcf    DPX+B3             ; the last bit is known to be
0239 191A                              rrcf    DPX+B2
023A 1919                              rrcf    DPX+B1
023B 1918                              rrcf    DPX+B0
                                 endif

023C 0002                    return

                        ;*****************************************************************************

                                       include "traject.asm"  ;
                        Trajectory Generation
                        ;*****************************************************************************
                        ;
                        ;                                Trajectory Generation Routines
                        ;
                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:          doPreMove
                        ;
                        ; DESCRIPTION:

                        doPreMove:

                        CLR16   INTEGRAL

023D 2996                       CLRF    INTEGRAL+B0
023E 2997                       CLRF    INTEGRAL+B1

                        MOV24   NMOVVAL,MOVVAL                  ; move buffer to MOVVAL

023F 6A5B                       MOVFP   NMOVVAL+B0,wreg         ; get byte of NMOVVAL into w
0240 4A5F                       MOVPF   wreg,MOVVAL+B0          ; move to MOVVAL(B0)
0241 6A5C                       MOVFP   NMOVVAL+B1,wreg         ; get byte of NMOVVAL into w
0242 4A60                       MOVPF   wreg,MOVVAL+B1          ; move to MOVVAL(B1)
0243 6A5D                       MOVFP   NMOVVAL+B2,wreg         ; get byte of NMOVVAL into w
0244 4A61                       MOVPF   wreg,MOVVAL+B2          ; move to MOVVAL(B2)

0245 8F93                       bcf     MOVSTAT,bit7            ; clear buffer flag
0246 8693                       bsf     MOVSTAT,bit6            ; set motion status flag
0247 8593                       bsf     MOVSTAT,bit5            ; set move in progress flag
0248 6AC2                       movfp   ONE,wreg
0249 4A94                       movpf   wreg,MOVFLAG            ; initialize MOVEFLAG to 1


024A 2951                       clrf    OPOSITION+B0            ; initialize buffers
                        MOV24   POSITION,OPOSITION+B1

024B 6A55                       MOVFP   POSITION+B0,wreg        ; get byte of POSITION into w
024C 4A52                       MOVPF   wreg,OPOSITION+B1+B0    ; move to OPOSITION+B1(B0)
024D 6A56                       MOVFP   POSITION+B1,wreg        ; get byte of POSITION into w
```


© 1993 Microchip Technology Inc.

```
024E 4A53                MOVPF   wreg,OPOSITION+B1+B1      ; move to OPOSITION+B1(B1)
024F 6A57                MOVFP   POSITION+B2,wreg          ; get byte of POSITION into w
0250 4A54                MOVPF   wreg,OPOSITION+B1+B2      ; move to OPOSITION+B1(B2)

                  MOV32  OPOSITION,MOVPBUF

0251 6A51                MOVFP   OPOSITION+B0,wreg         ; get byte of OPOSITION into w
0252 4AA4                MOVPF   wreg,MOVPBUF+B0           ; move to MOVPBUF(B0)
0253 6A52                MOVFP   OPOSITION+B1,wreg         ; get byte of OPOSITION into w
0254 4AA5                MOVPF   wreg,MOVPBUF+B1           ; move to MOVPBUF(B1)
0255 6A53                MOVFP   OPOSITION+B2,wreg         ; get byte of OPOSITION into w
0256 4AA6                MOVPF   wreg,MOVPBUF+B2           ; move to MOVPBUF(B2)
0257 6A54                MOVFP   OPOSITION+B3,wreg         ; get byte of OPOSITION into w
0258 4AA7                MOVPF   wreg,MOVPBUF+B3           ; move to MOVPBUF(B3)

0259 2995                clrf    SATFLAG
                  CLR16  MOVTIME                          ; clear move times

025A 2967                CLRF    MOVTIME+B0
025B 2968                CLRF    MOVTIME+B1

                  CLR16  T1                               ; 0 used as flag for no maximum
025C 296A                CLRF    T1+B0
025D 296B                CLRF    T1+B1

                  CLR16  T2
025E 296C                CLRF    T2+B0
025F 296D                CLRF    T2+B1

                  CLR16  TAU
0260 296E                CLRF    TAU+B0
0261 296F                CLRF    TAU+B1

                  CLR32  MOVDEL                           ; clear move discretization error

0262 29B0                CLRF    MOVDEL+B0
0263 29B1                CLRF    MOVDEL+B1
0264 29B2                CLRF    MOVDEL+B2
0265 29B3                CLRF    MOVDEL+B3

                   CLR16  PH2FLAT                         ; clear phase 2 flat counter
0266 29B4                CLRF    PH2FLAT+B0
0267 29B5                CLRF    PH2FLAT+B1


0268 3391                tstfsz  MODETYPE
0269 C2C5                goto    vmode
                  pmode
                  MOVFP24  MOVVAL,TMP
026A 785F                MOVFP   MOVVAL+B0,TMP+B0          ; move MOVVAL(B0) to TMP(B0)
026B 7960                MOVFP   MOVVAL+B1,TMP+B1          ; move MOVVAL(B1) to TMP(B1)
026C 7A61                MOVFP   MOVVAL+B2,TMP+B2          ; move MOVVAL(B2) to TMP(B2)

026D 971A                btfss   TMP+B2,MSB
026E C276                goto    mvpos
```


© 1993 Microchip Technology Inc.

```
                    NEG24   TMP
026F 1318                   COMF    TMP+B0
0270 1319                   COMF    TMP+B1
0271 131A                   COMF    TMP+B2
0272 290A                   CLRF    wreg
0273 1518                   INCF    TMP+B0
0274 1119                   ADDWFC  TMP+B1
0275 111A                   ADDWFC  TMP+B2

                    mvpos
0276 291C                   clrf    MOVTMP+B0           ; calculate  abs(MOVVAL) - 3
0277 291D                   clrf    MOVTMP+B1           ; do immediate move if nega-
tive
0278 291E                   clrf    MOVTMP+B2
0279 801C                   bsf     MOVTMP+B0,bit0
027A 811C                   bsf     MOVTMP+B0,bit1
                    SUB24   MOVTMP,TMP

027B 6A1C                   MOVFP   MOVTMP+B0,wreg      ; get lowest byte of MOVTMP
027C 0518                   SUBWF   TMP+B0              ; sub lowest byte of TMP, save
027D 6A1D                   MOVFP   MOVTMP+B1,wreg      ; get 2nd byte of MOVTMP into
027E 0319                   SUBWFB  TMP+B1              ; sub 2nd byte of TMP, save in
027F 6A1E                   MOVFP   MOVTMP+B2,wreg      ; get 3rd byte of MOVTMP into
0280 031A                   SUBWFB  TMP+B2              ; sub 3rd byte of TMP, save in



0281 971A                   btfss   TMP+B2,MSB          ; check for zero move
0282 C28E                   goto    nonzero
0283 2B90                   setf    SERVOFLAG           ; set servoflag to restore
0284 2994                   clrf    MOVFLAG
0285 8D93                   bcf     MOVSTAT,bit5
0286 8E93                   bcf     MOVSTAT,bit6
                    ADD24   MOVVAL,POSITION

0287 6A5F                   MOVFP   MOVVAL+B0,wreg      ; get lowest byte of MOVVAL
0288 0F55                   ADDWF   POSITION+B0         ; add lowest byte of POSITION,
0289 6A60                   MOVFP   MOVVAL+B1,wreg      ; get 2nd byte of MOVVAL into
028A 1156                   ADDWFC  POSITION+B1         ; add 2nd byte of POSITION,
028B 6A61                   MOVFP   MOVVAL+B2,wreg      ; get 3rd byte of MOVVAL into
028C 1157                   ADDWFC  POSITION+B2         ; add 3rd byte of POSITION,

028D 0002           return
                    nonzero
                    CLR32   MOVVBUF

028E 29A8                   CLRF    MOVVBUF+B0
028F 29A9                   CLRF    MOVVBUF+B1
0290 29AA                   CLRF    MOVVBUF+B2
0291 29AB                   CLRF    MOVVBUF+B3



0292 6A61                   movfp   MOVVAL+B2,wreg      ; move sign
0293 B580                   andlw   0x80
0294 4A69                   movpf   wreg,MOVSIGN

0295 29A3                   clrf    V+B3                ; create appropriate velocity
                    MOV24   VL,V                        ; acceleration limits from
0296 6A20                   MOVFP   VL+B0,wreg          ; get byte of VL into w
0297 4AA0                   MOVPF   wreg,V+B0           ; move to V(B0)
0298 6A21                   MOVFP   VL+B1,wreg          ; get byte of VL into w
0299 4AA1                   MOVPF   wreg,V+B1           ; move to V(B1)
029A 6A22                   MOVFP   VL+B2,wreg          ; get byte of VL into w
029B 4AA2                   MOVPF   wreg,V+B2           ; move to V(B2)
```


© 1993 Microchip Technology Inc.

```
029C 299F                clrf    A+B3
             MOV24       AL,A
029D 6A23                MOVFP   AL+B0,wreg          ; get byte of AL into w
029E 4A9C                MOVPF   wreg,A+B0           ; move to A(B0)
029F 6A24                MOVFP   AL+B1,wreg          ; get byte of AL into w
02A0 4A9D                MOVPF   wreg,A+B1           ; move to A(B1)
02A1 6A25                MOVFP   AL+B2,wreg          ; get byte of AL into w
02A2 4A9E                MOVPF   wreg,A+B2           ; move to A(B2)


02A3 290A                clrf    wreg
02A4 3269                cpfsgt  MOVSIGN
02A5 C2B8                goto    minc
             NEG32       V
02A6 13A0                COMF    V+B0
02A7 13A1                COMF    V+B1
02A8 13A2                COMF    V+B2
02A9 13A3                COMF    V+B3
02AA 290A                CLRF    wreg
02AB 15A0                INCF    V+B0
02AC 11A1                ADDWFC  V+B1
02AD 11A2                ADDWFC  V+B2
02AE 11A3                ADDWFC  V+B3

             NEG32       A
02AF 139C                COMF    A+B0
02B0 139D                COMF    A+B1
02B1 139E                COMF    A+B2
02B2 139F                COMF    A+B3
02B3 290A                CLRF    wreg
02B4 159C                INCF    A+B0
02B5 119D                ADDWFC  A+B1
02B6 119E                ADDWFC  A+B2
02B7 119F                ADDWFC  A+B3

             minc
02B8 2963                clrf    HMOVVAL+B0          ; evaluate MOVVAL/2
                         MOV24   MOVVAL,HMOVVAL+B1
02B9 6A5F                MOVFP   MOVVAL+B0,wreg      ; get byte of MOVVAL into w
02BA 4A64                MOVPF   wreg,HMOVVAL+B1+B0  ; move to HMOVVAL+B1(B0)
02BB 6A60                MOVFP   MOVVAL+B1,wreg      ; get byte of MOVVAL into w
02BC 4A65                MOVPF   wreg,HMOVVAL+B1+B1  ; move to HMOVVAL+B1(B1)
02BD 6A61                MOVFP   MOVVAL+B2,wreg      ; get byte of MOVVAL into w
02BE 4A66                MOVPF   wreg,HMOVVAL+B1+B2  ; move to HMOVVAL+B1(B2)

             RRC32       HMOVVAL                     ; half move in Q8
02BF 1A66                RLCF    HMOVVAL+B3,W        ; move sign into carry bit
02C0 1966                RRCF    HMOVVAL+B3
02C1 1965                RRCF    HMOVVAL+B2
02C2 1964                RRCF    HMOVVAL+B1
02C3 1963                RRCF    HMOVVAL+B0

02C4 C2FE                goto    modeready

             vmode
02C5 9F91                btfsc   MODETYPE,MSB        ; is it torque move?
02C6 C306                goto    tmode
```


© 1993 Microchip Technology Inc.

```
02C7 2966                clrf    HMOVVAL+B3                  ; compute final minus initial
                  MOV24  MOVVAL,HMOVVAL

02C8 6A5F                MOVFP   MOVVAL+B0,wreg              ; get byte of MOVVAL into w
02C9 4A63                MOVPF   wreg,HMOVVAL+B0             ; move to HMOVVAL(B0)
02CA 6A60                MOVFP   MOVVAL+B1,wreg              ; get byte of MOVVAL into w
02CB 4A64                MOVPF   wreg,HMOVVAL+B1             ; move to HMOVVAL(B1)
02CC 6A61                MOVFP   MOVVAL+B2,wreg              ; get byte of MOVVAL into w
02CD 4A65                MOVPF   wreg,HMOVVAL+B2             ; move to HMOVVAL(B2)

02CE 9F61                btfsc   MOVVAL+B2,MSB
02CF 2B66                setf    HMOVVAL+B3
                  SUB32  MOVVBUF,HMOVVAL

02D0 6AA8                MOVFP   MOVVBUF+B0,wreg     ; get lowest byte of MOVVBUF into w
02D1 0563                SUBWF   HMOVVAL+B0          ; sub lowest byte of HMOVVAL, save in
02D2 6AA9                MOVFP   MOVVBUF+B1,wreg     ; get 2nd byte of MOVVBUF into w
02D3 0364                SUBWFB  HMOVVAL+B1          ; sub 2nd byte of HMOVVAL, save in
02D4 6AAA                MOVFP   MOVVBUF+B2,wreg     ; get 3rd byte of MOVVBUF into w
02D5 0365                SUBWFB  HMOVVAL+B2          ; sub 3rd byte of HMOVVAL, save in
02D6 6AAB                MOVFP   MOVVBUF+B3,wreg     ; get 4th byte of MOVVBUF into w
02D7 0366                SUBWFB  HMOVVAL+B3          ; sub 4th byte of HMOVVAL, save in
02D8 6A66                movfp   HMOVVAL+B3,wreg
02D9 B580                andlw   0x80
02DA 4A69                movpf   wreg,MOVSIGN

02DB 29A3                clrf    V+B3                ; create appropriate velocity and
                  MOV24  VL,V                        ; acceleration limits from move sign

02DC 6A20                MOVFP   VL+B0,wreg          ; get byte of VL into w
02DD 4AA0                MOVPF   wreg,V+B0           ; move to V(B0)
02DE 6A21                MOVFP   VL+B1,wreg          ; get byte of VL into w
02DF 4AA1                MOVPF   wreg,V+B1           ; move to V(B1)
02E0 6A22                MOVFP   VL+B2,wreg          ; get byte of VL into w
02E1 4AA2                MOVPF   wreg,V+B2           ; move to V(B2)

02E2 299F                clrf    A+B3
                  MOV24  AL,A

02E3 6A23                MOVFP   AL+B0,wreg          ; get byte of AL into w
02E4 4A9C                MOVPF   wreg,A+B0           ; move to A(B0)
02E5 6A24                MOVFP   AL+B1,wreg          ; get byte of AL into w
02E6 4A9D                MOVPF   wreg,A+B1           ; move to A(B1)
02E7 6A25                MOVFP   AL+B2,wreg          ; get byte of AL into w
02E8 4A9E                MOVPF   wreg,A+B2           ; move to A(B2)

02E9 290A                clrf    wreg
02EA 3269                cpfsgt  MOVSIGN
02EB C2FE                goto    modeready
                  NEG32  V

02EC 13A0                COMF    V+B0
02ED 13A1                COMF    V+B1
02EE 13A2                COMF    V+B2
02EF 13A3                COMF    V+B3
02F0 290A                CLRF    wreg
02F1 15A0                INCF    V+B0
02F2 11A1                ADDWFC  V+B1
02F3 11A2                ADDWFC  V+B2
02F4 11A3                ADDWFC  V+B3

                  NEG32  A

02F5 139C                COMF    A+B0
02F6 139D                COMF    A+B1
```


© 1993 Microchip Technology Inc.

```
02F7 139E                       COMF    A+B2
02F8 139F                       COMF    A+B3
02F9 290A                       CLRF    wreg
02FA 159C                       INCF    A+B0
02FB 119D                       ADDWFC  A+B1
02FC 119E                       ADDWFC  A+B2
02FD 119F                       ADDWFC  A+B3
                                modeready
02FE 2962                       clrf    MOVVAL+B3
02FF 9F61                       btfsc   MOVVAL+B2,MSB
0300 2B62                       setf    MOVVAL+B3

0301 2B90                       setf    SERVOFLAG           ; set servoflag to restore servo
                                                            ; if stopped

                                if   _PICMASTER_DEBUG

                        ;***************************************************************
                        ;       For PICMASTER Debug/servo tuning puporses only Purposes Only
                        ;
                        testCapCount
0302 6ABC                       movfp   CAPCOUNT+B0,wreg
0303 08BD                       iorwf   CAPCOUNT+B1,W
0304 4ABB                       movpf   wreg,CAPFLAG
                        ;*******************************************************************************
                         endif

0305 0002                        return

                          tmode
                        MOV16   MOVVAL+B1,YPWM                  ; torque/voltage mode
                                                                ; set new commanded value

0306 6A60                        MOVFP   MOVVAL+B1+B0,wreg      ; get byte of MOVVAL+B1 into w
0307 0188                        MOVWF   YPWM+B0                ; move to YPWM(B0)
0308 6A61                        MOVFP   MOVVAL+B1+B1,wreg      ; get byte of MOVVAL+B1 into w
0309 0189                        MOVWF   YPWM+B1                ; move to YPWM(B1)


030A 2990                        clrf    SERVOFLAG              ; disable servo
030B E50B                        call    doTorque               ; set pwm duty cycle
030C 2994                        clrf    MOVFLAG
030D 8D93                        bcf     MOVSTAT,bit5
                          if     _PICMASTER_DEBUG
030E C302                        goto    testCapCount
                          else
                                 return
                          endif

                        ;*******************************************************************************

                        ;*******************************************************************************
                        ; NAME:          doMove
                        ;
                        ; DESCRIPTION:   In position mode, trapezoidal moves are performed. Phase1
                        ;               and phase2 respectively, are the periods for the first and
                        ;               second halves of the move. The move time is defined as zero
                        ;               at the beginning of the move,T2 is the time at half the
                                                        move, T1 is the time when c
                        ;               begins,(the region of constant velocity reduces to a point
                        ;               in the case where maximum speed is not realized, and the
                        ;               trapezoidal move degenerates into a trianglular move,
                        ;               together with T1=T2), and TAU is the total time of the move.
                        ;               The accelerations are +-AL or 0.
                        ;
                        ;
                        ;                triangle speed                 trapezoidal speed
                        ;
                        ;                                                ________________
                        ;                                              /                \
```

```
                ;                /\                                  /                    \
                ;               /  \                                /                      \
                ;              /    \                              /                        \
                ;             /      \                            /                          \
                ;            /        \                          /                            \
                ;
                ;          0   T1=T2  TAU                        0    T1       T2             TAU
                ;
                ;
                ;                 Let x denote the undershoot and y the overshoot commanded
                ;                 at adjacent sample times as half the move is crossed.
                ;                 In the case of a triangular move, the discretization error
                ;                 is given by
                ;
                ;                          error = min (2x,2y)
                ;
                ;                 For a trapezoidal move, the discretization error is
                ;
                ;                        error = min (2x,y-x) <= .5*(maximum commanded speed)
                ;
                ;                 This discretization error is resolved in the final sample
                ;                 time of the move by executing a step to the final position
                ;                 at zero speed. The method employed here the best possible
                ;                 performance with regard to discretization error without
                ;                 dynamically modifying velocity and acceleration limits.
                ;
                ;
                ;
                ;                 In velocity mode, ramp moves are performed.
                ;
                ;                                                    / final velocity
                ;                                                   /
                ;                                                  /
                ;                                                 /
                ;                                                /
                ;                                               /
                ;                                              /
                ;                        initial velocity     /
                ;                                             0      TAU
                ;
                ;
                doMove

                 INC16   MOVTIME                          ; increment move time

030F 290A                CLRF    wreg
0310 1567                INCF    (MOVTIME)+B0
0311 1168                ADDWFC  (MOVTIME)+B1

0312 E468                call    doPosVel                ; evaluate iterative equations
0313 3391                tstfsz  MODETYPE
0314 C429                goto    vmove
                         pmove
0315 6AC2                movfp   ONE,wreg                 ; test if in phase1
0316 3194                cpfseq  MOVFLAG
0317 C3DB                goto    phase2
                  phase1
                         MOVFP32  MOVDEL,MOVTMP          ; save previous discretization error
0318 7CB0                MOVFP   MOVDEL+B0,MOVTMP+B0     ; move MOVDEL(B0) to MOVTMP(B0)
0319 7DB1                MOVFP   MOVDEL+B1,MOVTMP+B1     ; move MOVDEL(B1) to MOVTMP(B1)
031A 7EB2                MOVFP   MOVDEL+B2,MOVTMP+B2     ; move MOVDEL(B2) to MOVTMP(B2)
031B 7FB3                MOVFP   MOVDEL+B3,MOVTMP+B3     ; move MOVDEL(B3) to MOVTMP(B3)
                         MOV32   OPOSITION,MOVDEL        ; test if half move
031C 6A51                MOVFP   OPOSITION+B0,wreg       ; get byte of OPOSITION into w
031D 4AB0                MOVPF   wreg,MOVDEL+B0          ; move to MOVDEL(B0)
031E 6A52                MOVFP   OPOSITION+B1,wreg       ; get byte of OPOSITION into w
031F 4AB1                MOVPF   wreg,MOVDEL+B1          ; move to MOVDEL(B1)
```


© 1993 Microchip Technology Inc.

```
0320 6A53                MOVFP   OPOSITION+B2,wreg      ; get byte of OPOSITION into w
0321 4AB2                MOVPF   wreg,MOVDEL+B2         ; move to MOVDEL(B2)
0322 6A54                MOVFP   OPOSITION+B3,wreg      ; get byte of OPOSITION into w
0323 4AB3                MOVPF   wreg,MOVDEL+B3         ; move to MOVDEL(B3)
                         ADD32   HMOVVAL,MOVDEL
0324 6A63                MOVFP   HMOVVAL+B0,wreg        ; get lowest byte of HMOVVAL into w
0325 0FB0                ADDWF   MOVDEL+B0              ; add lowest byte of MOVDEL, save in
0326 6A64                MOVFP   HMOVVAL+B1,wreg        ; get 2nd byte of HMOVVAL into w
0327 11B1                ADDWFC  MOVDEL+B1              ; add 2nd byte of MOVDEL, save in
0328 6A65                MOVFP   HMOVVAL+B2,wreg        ; get 3rd byte of HMOVVAL into w
0329 11B2                ADDWFC  MOVDEL+B2              ; add 3rd byte of MOVDEL, save in
032A 6A66                MOVFP   HMOVVAL+B3,wreg        ; get 4th byte of HMOVVAL into w
032B 11B3                ADDWFC  MOVDEL+B3              ; add 4th byte of MOVDEL, save in
                         SUB32   MOVPBUF,MOVDEL
032C 6AA4                MOVFP   MOVPBUF+B0,wreg        ; get lowest byte of MOVPBUF into
032D 05B0                SUBWF   MOVDEL+B0              ; sub lowest byte of MOVDEL, save
032E 6AA5                MOVFP   MOVPBUF+B1,wreg        ; get 2nd byte of MOVPBUF into w
032F 03B1                SUBWFB  MOVDEL+B1              ; sub 2nd byte of MOVDEL, save in
0330 6AA6                MOVFP   MOVPBUF+B2,wreg        ; get 3rd byte of MOVPBUF into w
0331 03B2                SUBWFB  MOVDEL+B2              ; sub 3rd byte of MOVDEL, save in
0332 6AA7                MOVFP   MOVPBUF+B3,wreg        ; get 4th byte of MOVPBUF into w
0333 03B3                SUBWFB  MOVDEL+B3              ; sub 4th byte of MOVDEL, save in
0334 9769                btfss   MOVSIGN,MSB
0335 C33F                goto    mpos1
                         NEG32   MOVDEL
0336 13B0                COMF    MOVDEL+B0
0337 13B1                COMF    MOVDEL+B1
0338 13B2                COMF    MOVDEL+B2
0339 13B3                COMF    MOVDEL+B3
033A 290A                CLRF    wreg
033B 15B0                INCF    MOVDEL+B0
033C 11B1                ADDWFC  MOVDEL+B1
033D 11B2                ADDWFC  MOVDEL+B2
033E 11B3                ADDWFC  MOVDEL+B3
                         mpos1
033F 97B3                btfss   MOVDEL+B3,MSB
0340 C3A5                goto    speedup                ; continue to speed up if in
                         TFSZ16  T1                     ; if T1=0, maximum velocity not
0341 6A6A                MOVFP   T1+B0,wreg
0342 086B                IORWF   T1+B1,W
0343 330A                TSTFSZ  wreg
                                                        ; reached,
                                                        ; has been set in speedup
0344 C378                goto    t2net1
                         NEG32   A                      ; negate A for speeddown
0345 139C                COMF    A+B0
0346 139D                COMF    A+B1
0347 139E                COMF    A+B2
0348 139F                COMF    A+B3
0349 290A                CLRF    wreg
034A 159C                INCF    A+B0
034B 119D                ADDWFC  A+B1
034C 119E                ADDWFC  A+B2
034D 119F                ADDWFC  A+B3
                         ADD32   MOVDEL,MOVTMP          ; test x-y < 0
034E 6AB0                MOVFP   MOVDEL+B0,wreg         ; get lowest byte of MOVDEL into
034F 0F1C                ADDWF   MOVTMP+B0              ; add lowest byte of MOVTMP, save
0350 6AB1                MOVFP   MOVDEL+B1,wreg         ; get 2nd byte of MOVDEL into w
0351 111D                ADDWFC  MOVTMP+B1              ; add 2nd byte of MOVTMP, save in
0352 6AB2                MOVFP   MOVDEL+B2,wreg         ; get 3rd byte of MOVDEL into w
0353 111E                ADDWFC  MOVTMP+B2              ; add 3rd byte of MOVTMP, save in
0354 6AB3                MOVFP   MOVDEL+B3,wreg         ; get 4th byte of MOVDEL into w
0355 111F                ADDWFC  MOVTMP+B3              ; add 4th byte of MOVTMP, save in
0356 971F                btfss   MOVTMP+B3,MSB          ; if new discretization error larger,
0357 C36E                goto    triok                  ; backup to define T2, otherwise ok
0358 2B6C                setf    T2+B0                  ; set T2=-1 for backup
0359 2B6D                setf    T2+B1
                         NEG32   A                      ; negate A to undo
```

4

© 1993 Microchip Technology Inc.

```
035A 139C                    COMF    A+B0
035B 139D                    COMF    A+B1
035C 139E                    COMF    A+B2
035D 139F                    COMF    A+B3
035E 290A                    CLRF    wreg
035F 159C                    INCF    A+B0
0360 119D                    ADDWFC  A+B1
0361 119E                    ADDWFC  A+B2
0362 119F                    ADDWFC  A+B3
0363 E48A                    call    undoPosVel
                             NEG32   A                       ; negate A again for speeddown
0364 139C                    COMF    A+B0
0365 139D                    COMF    A+B1
0366 139E                    COMF    A+B2
0367 139F                    COMF    A+B3
0368 290A                    CLRF    wreg
0369 159C                    INCF    A+B0
036A 119D                    ADDWFC  A+B1
036B 119E                    ADDWFC  A+B2
036C 119F                    ADDWFC  A+B3
036D E468                    call    doPosVel                ; and reevaluate iterative equations
                               triok
                             ADD16   MOVTIME,T2                      ; add time to T2
036E 6A67                    MOVFP   MOVTIME+B0,wreg         ; get lowest byte of MOVTIME into w
036F 0F6C                    ADDWF   T2+B0                   ; add lowest byte of T2, save in
0370 6A68                    MOVFP   MOVTIME+B1,wreg         ; get 2nd byte of MOVTIME into w
0371 116D                    ADDWFC  T2+B1                   ; add 2nd byte of T2, save in T2(B1)
                             MOV16   T2,T1
0372 6A6C                    MOVFP   T2+B0,wreg              ; get byte of T2 into w
0373 016A                    MOVWF   T1+B0                   ; move to T1(B0)
0374 6A6D                    MOVFP   T2+B1,wreg              ; get byte of T2 into w
0375 016B                    MOVWF   T1+B1                   ; move to T1(B1)
0376 1594                    incf    MOVFLAG                 ; increment move flag for
0377 C3CE                    goto    mvok                    ; execute last phase1 move
                 t2net1
0378 2B6C                    setf    T2+B0                   ; set T2=-1 for backup
0379 2B6D                    setf    T2+B1
                             ADD16   MOVTIME,T2              ; add time to T2
037A 6A67                    MOVFP   MOVTIME+B0,wreg         ; get lowest byte of MOVTIME
037B 0F6C                    ADDWF   T2+B0                   ; add lowest byte of T2, save
037C 6A68                    MOVFP   MOVTIME+B1,wreg         ; get 2nd byte of MOVTIME into
037D 116D                    ADDWFC  T2+B1                   ; add 2nd byte of T2, save in T2(B1)
                             MOVFP32  MOVTMP,TMP             ; test if 3x-y < 0
037E 781C                    MOVFP   MOVTMP+B0,TMP+B0        ; move MOVTMP(B0) to TMP(B0)
037F 791D                    MOVFP   MOVTMP+B1,TMP+B1        ; move MOVTMP(B1) to TMP(B1)
0380 7A1E                    MOVFP   MOVTMP+B2,TMP+B2        ; move MOVTMP(B2) to TMP(B2)
0381 7B1F                    MOVFP   MOVTMP+B3,TMP+B3        ; move MOVTMP(B3) to TMP(B3)
                             RLC32   MOVTMP
0382 8804                    BCF     _carry
0383 1B1C                    RLCF    MOVTMP+B0
0384 1B1D                    RLCF    MOVTMP+B1
0385 1B1E                    RLCF    MOVTMP+B2
0386 1B1F                    RLCF    MOVTMP+B3
                             ADD32   TMP,MOVTMP
0387 6A18                    MOVFP   TMP+B0,wreg             ; get lowest byte of TMP into
0388 0F1C                    ADDWF   MOVTMP+B0               ; add lowest byte of MOVTMP,
0389 6A19                    MOVFP   TMP+B1,wreg             ; get 2nd byte of TMP into w
038A 111D                    ADDWFC  MOVTMP+B1               ; add 2nd byte of MOVTMP, save
038B 6A1A                    MOVFP   TMP+B2,wreg             ; get 3rd byte of TMP into w
038C 111E                    ADDWFC  MOVTMP+B2               ; add 3rd byte of MOVTMP, save
038D 6A1B                    MOVFP   TMP+B3,wreg             ; get 4th byte of TMP into w
038E 111F                    ADDWFC  MOVTMP+B3               ; add 4th byte of MOVTMP, save
                             ADD32   MOVDEL,MOVTMP
038F 6AB0                    MOVFP   MOVDEL+B0,wreg          ; get lowest byte of MOVDEL
0390 0F1C                    ADDWF   MOVTMP+B0               ; add lowest byte of MOVTMP,
0391 6AB1                    MOVFP   MOVDEL+B1,wreg          ; get 2nd byte of MOVDEL into
0392 111D                    ADDWFC  MOVTMP+B1               ; add 2nd byte of MOVTMP, save
```


© 1993 Microchip Technology Inc.

```
0393 6AB2                MOVFP   MOVDEL+B2,wreg      ; get 3rd byte of MOVDEL into
0394 111E                ADDWFC  MOVTMP+B2           ; add 3rd byte of MOVTMP, save
0395 6AB3                MOVFP   MOVDEL+B3,wreg      ; get 4th byte of MOVDEL into
0396 111F                ADDWFC  MOVTMP+B3           ; add 4th byte of MOVTMP, save
0397 971F                btfss   MOVTMP+B3,MSB       ; if new discretization error
0398 C39B                goto    trapok              ; take one more flat step
0399 2BB4                setf    PH2FLAT+B0
039A 2BB5                setf    PH2FLAT+B1
                   trapok
                         ADD16   T2,PH2FLAT
039B 6A6C                MOVFP   T2+B0,wreg          ; get lowest byte of T2 into w
039C 0FB4                ADDWF   PH2FLAT+B0          ; add lowest byte of PH2FLAT,
039D 6A6D                MOVFP   T2+B1,wreg          ; get 2nd byte of T2 into w
039E 11B5                ADDWFC  PH2FLAT+B1          ; add 2nd byte of PH2FLAT,
                         SUB16   T1,PH2FLAT
039F 6A6A                MOVFP   T1+B0,wreg          ; get lowest byte of T1 into w
03A0 05B4                SUBWF   PH2FLAT+B0          ; sub lowest byte of PH2FLAT,
03A1 6A6B                MOVFP   T1+B1,wreg          ; get 2nd byte of T1 into w
03A2 03B5                SUBWFB  PH2FLAT+B1          ; sub 2nd byte of PH2FLAT,
03A3 1594                incf    MOVFLAG             ; increment move flag for
03A4 C3CE                goto    mvok                ; execute last phase1 move
                         speedup
                          MOVFP32  V,MOVTMP          ; test if maximum velocity
03A5 7CA0                MOVFP   V+B0,MOVTMP+B0      ; move V(B0) to MOVTMP(B0)
03A6 7DA1                MOVFP   V+B1,MOVTMP+B1      ; move V(B1) to MOVTMP(B1)
03A7 7EA2                MOVFP   V+B2,MOVTMP+B2      ; move V(B2) to MOVTMP(B2)
03A8 7FA3                MOVFP   V+B3,MOVTMP+B3      ; move V(B3) to MOVTMP(B3)
                         SUB32   MOVVBUF,MOVTMP
03A9 6AA8                MOVFP   MOVVBUF+B0,wreg     ; get lowest byte of MOVVBUF
03AA 051C                SUBWF   MOVTMP+B0           ; sub lowest byte of MOVTMP,
03AB 6AA9                MOVFP   MOVVBUF+B1,wreg     ; get 2nd byte of MOVVBUF into
03AC 031D                SUBWFB  MOVTMP+B1           ; sub 2nd byte of MOVTMP, save
03AD 6AAA                MOVFP   MOVVBUF+B2,wreg     ; get 3rd byte of MOVVBUF into
03AE 031E                SUBWFB  MOVTMP+B2           ; sub 3rd byte of MOVTMP, save
03AF 6AAB                MOVFP   MOVVBUF+B3,wreg     ; get 4th byte of MOVVBUF into
03B0 031F                SUBWFB  MOVTMP+B3           ; sub 4th byte of MOVTMP, save in
03B1 9769                btfss   MOVSIGN,MSB
03B2 C3BC                goto    mpos
                         NEG32   MOVTMP
03B3 131C                COMF    MOVTMP+B0
03B4 131D                COMF    MOVTMP+B1
03B5 131E                COMF    MOVTMP+B2
03B6 131F                COMF    MOVTMP+B3
03B7 290A                CLRF    wreg
03B8 151C                INCF    MOVTMP+B0
03B9 111D                ADDWFC  MOVTMP+B1
03BA 111E                ADDWFC  MOVTMP+B2
03BB 111F                ADDWFC  MOVTMP+B3
                   mpos
03BC 971F                btfss   MOVTMP+B3,MSB
03BD C3CE                goto    mvok                ; if not, execute move
                         TFSZ16  T1                  ; if so, check to see if T1
03BE 6A6A                MOVFP   T1+B0,wreg
03BF 086B                IORWF   T1+B1,W
03C0 330A                TSTFSZ  wreg                ; already been set
03C1 C3CE                goto    mvok
03C2 E48A                call    undoPosVel          ; if not, backup and redo
                         CLR32   A                   ; equations, resulting in an
03C3 299C                CLRF    A+B0
03C4 299D                CLRF    A+B1
03C5 299E                CLRF    A+B2
03C6 299F                CLRF    A+B3
03C7 E468                call    doPosVel            ; maximum speed <= VL
03C8 2B6A                setf    T1+B0               ; evaluate T1
03C9 2B6B                setf    T1+B1
                         ADD16   MOVTIME,T1
03CA 6A67                MOVFP   MOVTIME+B0,wreg     ; get lowest byte of MOVTIME
03CB 0F6A                ADDWF   T1+B0               ; add lowest byte of T1, save
```


© 1993 Microchip Technology Inc.

```
03CC 6A68                    MOVFP   MOVTIME+B1,wreg      ; get 2nd byte of MOVTIME into
03CD 116B                    ADDWFC  T1+B1                ; add 2nd byte of T1, save in
                       mvok
                             MOV24   MOVPBUF+B1,POSITION  ; move Q8 calculated position
03CE 6AA5                    MOVFP   MOVPBUF+B1+B0,wreg   ; get byte of MOVPBUF+B1 into
03CF 4A55                    MOVPF   wreg,POSITION+B0     ; move to POSITION(B0)
03D0 6AA6                    MOVFP   MOVPBUF+B1+B1,wreg   ; get byte of MOVPBUF+B1 into
03D1 4A56                    MOVPF   wreg,POSITION+B1     ; move to POSITION(B1)
03D2 6AA7                    MOVFP   MOVPBUF+B1+B2,wreg   ; get byte of MOVPBUF+B1 into
03D3 4A57                    MOVPF   wreg,POSITION+B2     ; move to POSITION(B2)
                             MOV24   MOVVBUF+B0,VELOCITY  ; move Q0 calculated velocity
03D4 6AA8                    MOVFP   MOVVBUF+B0+B0,wreg   ; get byte of MOVVBUF+B0 into
03D5 4A58                    MOVPF   wreg,VELOCITY+B0     ; move to VELOCITY(B0)
03D6 6AA9                    MOVFP   MOVVBUF+B0+B1,wreg   ; get byte of MOVVBUF+B0 into
03D7 4A59                    MOVPF   wreg,VELOCITY+B1     ; move to VELOCITY(B1)
03D8 6AAA                    MOVFP   MOVVBUF+B0+B2,wreg   ; get byte of MOVVBUF+B0 into
03D9 4A5A                    MOVPF   wreg,VELOCITY+B2     ; move to VELOCITY(B2)
03DA 0002                    return

                             phase2
                             TFSZ16  PH2FLAT              ; is flat section finished?
03DB 6AB4                    MOVFP   PH2FLAT+B0,wreg
03DC 08B5                    IORWF   PH2FLAT+B1,W
03DD 330A                    TSTFSZ  wreg
03DE C3FF                    goto    flat
                             TFSZ32  MOVVBUF              ; is velocity zero?
03DF 6AA8                    MOVFP   MOVVBUF+B0,wreg
03E0 08A9                    IORWF   MOVVBUF+B1,W
03E1 08AA                    IORWF   MOVVBUF+B2,W
03E2 08AB                    IORWF   MOVVBUF+B3,W
03E3 330A                    TSTFSZ  wreg
03E4 C41C                    goto    mready               ; if not, execute move
03E5 2994                    clrf    MOVFLAG              ; if so, clear MOVFLAG
03E6 8E93                    bcf     MOVSTAT,bit6         ; clear motion status flag
03E7 8D93                    bcf     MOVSTAT,bit5         ; clear move in progress flag
                             CLR32   A                    ; set zero velocity and acceleration,
03E8 299C                    CLRF    A+B0
03E9 299D                    CLRF    A+B1
03EA 299E                    CLRF    A+B2
03EB 299F                    CLRF    A+B3
                             MOV16   MOVTIME,TAU
03EC 6A67                    MOVFP   MOVTIME+B0,wreg      ; get byte of MOVTIME into w
03ED 016E                    MOVWF   TAU+B0               ; move to TAU(B0)
03EE 6A68                    MOVFP   MOVTIME+B1,wreg      ; get byte of MOVTIME into w
03EF 016F                    MOVWF   TAU+B1               ; move to TAU(B1)
                             MOV32   OPOSITION,MOVPBUF    ; execute last move to P(0)+MOVVAL
03F0 6A51                    MOVFP   OPOSITION+B0,wreg    ; get byte of OPOSITION into w
03F1 4AA4                    MOVPF   wreg,MOVPBUF+B0      ; move to MOVPBUF(B0)
03F2 6A52                    MOVFP   OPOSITION+B1,wreg    ; get byte of OPOSITION into w
03F3 4AA5                    MOVPF   wreg,MOVPBUF+B1      ; move to MOVPBUF(B1)
03F4 6A53                    MOVFP   OPOSITION+B2,wreg    ; get byte of OPOSITION into w
03F5 4AA6                    MOVPF   wreg,MOVPBUF+B2      ; move to MOVPBUF(B2)
03F6 6A54                    MOVFP   OPOSITION+B3,wreg    ; get byte of OPOSITION into w
03F7 4AA7                    MOVPF   wreg,MOVPBUF+B3      ; move to MOVPBUF(B3)
                             ADD24   MOVVAL,MOVPBUF+B1
03F8 6A5F                    MOVFP   MOVVAL+B0,wreg       ; get lowest byte of MOVVAL into w
03F9 0FA5                    ADDWF   MOVPBUF+B1+B0        ; add lowest byte of MOVPBUF+B1, save
03FA 6A60                    MOVFP   MOVVAL+B1,wreg       ; get 2nd byte of MOVVAL into w
03FB 11A6                    ADDWFC  MOVPBUF+B1+B1        ; add 2nd byte of MOVPBUF+B1, save in
03FC 6A61                    MOVFP   MOVVAL+B2,wreg       ; get 3rd byte of MOVVAL into w
03FD 11A7                    ADDWFC  MOVPBUF+B1+B2        ; add 3rd byte of MOVPBUF+B1, save in
03FE C41C                    goto    mready

                       flat
03FF 2B1C                    setf    MOVTMP+B0
0400 2B1D                    setf    MOVTMP+B1
                             ADD16   MOVTMP,PH2FLAT       ; decrement by one use DEC16
0401 6A1C                    MOVFP   MOVTMP+B0,wreg       ; get lowest byte of MOVTMP into w
0402 0FB4                    ADDWF   PH2FLAT+B0           ; add lowest byte of PH2FLAT, save in
```


© 1993 Microchip Technology Inc.

```
0403 6A1D                    MOVFP   MOVTMP+B1,wreg  ; get 2nd byte of MOVTMP into w
0404 11B5                    ADDWFC  PH2FLAT+B1      ; add 2nd byte of PH2FLAT, save in
                             TFSZ16  PH2FLAT
0405 6AB4                    MOVFP   PH2FLAT+B0,wreg
0406 08B5                    IORWF   PH2FLAT+B1,W
0407 330A                    TSTFSZ  wreg
0408 C41C                    goto    mready
0409 299F                    clrf    A+B3            ; begin speed down section
                             MOV24   AL,A
040A 6A23                    MOVFP   AL+B0,wreg      ; get byte of AL into w
040B 4A9C                    MOVPF   wreg,A+B0       ; move to A(B0)
040C 6A24                    MOVFP   AL+B1,wreg      ; get byte of AL into w
040D 4A9D                    MOVPF   wreg,A+B1       ; move to A(B1)
040E 6A25                    MOVFP   AL+B2,wreg      ; get byte of AL into w
040F 4A9E                    MOVPF   wreg,A+B2       ; move to A(B2)
0410 290A                    clrf    wreg
0411 3169                    cpfseq  MOVSIGN
0412 C41C                    goto    mready
                             NEG32   A
0413 139C                    COMF    A+B0
0414 139D                    COMF    A+B1
0415 139E                    COMF    A+B2
0416 139F                    COMF    A+B3
0417 290A                    CLRF    wreg
0418 159C                    INCF    A+B0
0419 119D                    ADDWFC  A+B1
041A 119E                    ADDWFC  A+B2
041B 119F                    ADDWFC  A+B3
                mready
                             MOV24   MOVPBUF+B1,POSITION
041C 6AA5                    MOVFP   MOVPBUF+B1+B0,wreg  ; get byte of MOVPBUF+B1 into w
041D 4A55                    MOVPF   wreg,POSITION+B0    ; move to POSITION(B0)
041E 6AA6                    MOVFP   MOVPBUF+B1+B1,wreg  ; get byte of MOVPBUF+B1 into w
041F 4A56                    MOVPF   wreg,POSITION+B1    ; move to POSITION(B1)
0420 6AA7                    MOVFP   MOVPBUF+B1+B2,wreg  ; get byte of MOVPBUF+B1 into w
0421 4A57                    MOVPF   wreg,POSITION+B2    ; move to POSITION(B2)
                             MOV24   MOVVBUF+B0,VELOCITY
0422 6AA8                    MOVFP   MOVVBUF+B0+B0,wreg  ; get byte of MOVVBUF+B0 into w
0423 4A58                    MOVPF   wreg,VELOCITY+B0    ; move to VELOCITY(B0)
0424 6AA9                    MOVFP   MOVVBUF+B0+B1,wreg  ; get byte of MOVVBUF+B0 into w
0425 4A59                    MOVPF   wreg,VELOCITY+B1    ; move to VELOCITY(B1)
0426 6AAA                    MOVFP   MOVVBUF+B0+B2,wreg  ; get byte of MOVVBUF+B0 into w
0427 4A5A                    MOVPF   wreg,VELOCITY+B2    ; move to VELOCITY(B2)
0428 0002                    return
                vmove
                             MOVFP32  MOVVAL,MOVTMP      ; test if final velocity reached
0429 7C5F                    MOVFP   MOVVAL+B0,MOVTMP+B0 ; move MOVVAL(B0) to MOVTMP(B0)
042A 7D60                    MOVFP   MOVVAL+B1,MOVTMP+B1 ; move MOVVAL(B1) to MOVTMP(B1)
042B 7E61                    MOVFP   MOVVAL+B2,MOVTMP+B2 ; move MOVVAL(B2) to MOVTMP(B2)
042C 7F62                    MOVFP   MOVVAL+B3,MOVTMP+B3 ; move MOVVAL(B3) to MOVTMP(B3)
                             SUB32   MOVVBUF,MOVTMP
042D 6AA8                    MOVFP   MOVVBUF+B0,wreg     ; get lowest byte of MOVVBUF into w
042E 051C                    SUBWF   MOVTMP+B0           ; sub lowest byte of MOVTMP, save in
042F 6AA9                    MOVFP   MOVVBUF+B1,wreg     ; get 2nd byte of MOVVBUF into w
0430 031D                    SUBWFB  MOVTMP+B1           ; sub 2nd byte of MOVTMP, save in
0431 6AAA                    MOVFP   MOVVBUF+B2,wreg     ; get 3rd byte of MOVVBUF into w
0432 031E                    SUBWFB  MOVTMP+B2           ; sub 3rd byte of MOVTMP, save in
0433 6AAB                    MOVFP   MOVVBUF+B3,wreg     ; get 4th byte of MOVVBUF into w
0434 031F                    SUBWFB  MOVTMP+B3           ; sub 4th byte of MOVTMP, save in
0435 9769                    btfss   MOVSIGN,MSB
0436 C440                    goto    vmpos
                             NEG32   MOVTMP
0437 131C                    COMF    MOVTMP+B0
0438 131D                    COMF    MOVTMP+B1
0439 131E                    COMF    MOVTMP+B2
043A 131F                    COMF    MOVTMP+B3
043B 290A                    CLRF    wreg
043C 151C                    INCF    MOVTMP+B0
```

4

© 1993 Microchip Technology Inc.

```
043D 111D                ADDWFC  MOVTMP+B1
043E 111E                ADDWFC  MOVTMP+B2
043F 111F                ADDWFC  MOVTMP+B3
            vmpos
0440 971F                btfss   MOVTMP+B3,MSB
0441 C45B                goto    vmoveok             ; if not, continue
                         CLR32   A                   ; if so, set A=0 and continue with
0442 299C                CLRF    A+B0
0443 299D                CLRF    A+B1
0444 299E                CLRF    A+B2
0445 299F                CLRF    A+B3
                         MOV32   MOVVAL,MOVVBUF      ; move unless the final velocity
0446 6A5F                MOVFP   MOVVAL+B0,wreg      ; get byte of MOVVAL into w
0447 4AA8                MOVPF   wreg,MOVVBUF+B0     ; move to MOVVBUF(B0)
0448 6A60                MOVFP   MOVVAL+B1,wreg      ; get byte of MOVVAL into w
0449 4AA9                MOVPF   wreg,MOVVBUF+B1     ; move to MOVVBUF(B1)
044A 6A61                MOVFP   MOVVAL+B2,wreg      ; get byte of MOVVAL into w
044B 4AAA                MOVPF   wreg,MOVVBUF+B2     ; move to MOVVBUF(B2)
044C 6A62                MOVFP   MOVVAL+B3,wreg      ; get byte of MOVVAL into w
044D 4AAB                MOVPF   wreg,MOVVBUF+B3     ; move to MOVVBUF(B3)
                                                     ; is zero.
044E 2994                clrf    MOVFLAG             ; clear MOVFLAG
044F 8D93                bcf     MOVSTAT,bit5        ; clear move in progress flag
                         MOV16   MOVTIME,TAU
0450 6A67                MOVFP   MOVTIME+B0,wreg     ; get byte of MOVTIME into w
0451 016E                MOVWF   TAU+B0              ; move to TAU(B0)
0452 6A68                MOVFP   MOVTIME+B1,wreg     ; get byte of MOVTIME into w
0453 016F                MOVWF   TAU+B1              ; move to TAU(B1)
                         TFSZ32  MOVVAL
0454 6A5F                MOVFP   MOVVAL+B0,wreg
0455 0860                IORWF   MOVVAL+B1,W
0456 0861                IORWF   MOVVAL+B2,W
0457 0862                IORWF   MOVVAL+B3,W
0458 330A                TSTFSZ  wreg
0459 C45B                goto    vmoveok
045A 8E93                bcf     MOVSTAT,bit6        ; if final velocity is zero, clear
                                                     ; motion status flag
            vmoveok
                         MOV24   MOVPBUF+B1,POSITION
045B 6AA5                MOVFP   MOVPBUF+B1+B0,wreg  ; get byte of MOVPBUF+B1 into w
045C 4A55                MOVPF   wreg,POSITION+B0    ; move to POSITION(B0)
045D 6AA6                MOVFP   MOVPBUF+B1+B1,wreg  ; get byte of MOVPBUF+B1 into w
045E 4A56                MOVPF   wreg,POSITION+B1    ; move to POSITION(B1)
045F 6AA7                MOVFP   MOVPBUF+B1+B2,wreg  ; get byte of MOVPBUF+B1 into w
0460 4A57                MOVPF   wreg,POSITION+B2    ; move to POSITION(B2)
                         MOV24   MOVVBUF+B0,VELOCITY
0461 6AA8                MOVFP   MOVVBUF+B0+B0,wreg  ; get byte of MOVVBUF+B0 into w
0462 4A58                MOVPF   wreg,VELOCITY+B0    ; move to VELOCITY(B0)
0463 6AA9                MOVFP   MOVVBUF+B0+B1,wreg  ; get byte of MOVVBUF+B0 into w
0464 4A59                MOVPF   wreg,VELOCITY+B1    ; move to VELOCITY(B1)
0465 6AAA                MOVFP   MOVVBUF+B0+B2,wreg  ; get byte of MOVVBUF+B0 into w
0466 4A5A                MOVPF   wreg,VELOCITY+B2    ; move to VELOCITY(B2)
0467 0002                return
```


© 1993 Microchip Technology Inc.

```
                ;*****************************************************************
                ;*****************************************************************
                ; NAME:         doPosVel
                ;
                ; DESCRIPTION:  Evaluates the iterative equations for trapezoidal
                ;               generation
                ;
                ;               V(k)=V(k-1)+A,          P(k)=P(k-1)+V(k-1)+A/2,
                ;
                ;               where abs(A)={AL,0} depending on the region of the
                ;               being executed.
                ;
                doPosVel

                        ADD32   MOVVBUF,MOVPBUF         ; P(k-1)+V(k-1)
0468 6AA8               MOVFP   MOVVBUF+B0,wreg         ; get lowest byte of MOVVBUF into w
0469 0FA4               ADDWF   MOVPBUF+B0              ; add lowest byte of MOVPBUF, save in
046A 6AA9               MOVFP   MOVVBUF+B1,wreg         ; get 2nd byte of MOVVBUF into w
046B 11A5               ADDWFC  MOVPBUF+B1              ; add 2nd byte of MOVPBUF, save in
046C 6AAA               MOVFP   MOVVBUF+B2,wreg         ; get 3rd byte of MOVVBUF into w
046D 11A6               ADDWFC  MOVPBUF+B2              ; add 3rd byte of MOVPBUF, save in
046E 6AAB               MOVFP   MOVVBUF+B3,wreg         ; get 4th byte of MOVVBUF into w
046F 11A7               ADDWFC  MOVPBUF+B3              ; add 4th byte of MOVPBUF, save in

                        ADD32   A,MOVVBUF                       ; V(k)=V(k-1)+A
0470 6A9C               MOVFP   A+B0,wreg               ; get lowest byte of A into w
0471 0FA8               ADDWF   MOVVBUF+B0              ; add lowest byte of MOVVBUF, save in
0472 6A9D               MOVFP   A+B1,wreg               ; get 2nd byte of A into w
0473 11A9               ADDWFC  MOVVBUF+B1              ; add 2nd byte of MOVVBUF, save in
0474 6A9E               MOVFP   A+B2,wreg               ; get 3rd byte of A into w
0475 11AA               ADDWFC  MOVVBUF+B2              ; add 3rd byte of MOVVBUF, save in
0476 6A9F               MOVFP   A+B3,wreg               ; get 4th byte of A into w
0477 11AB               ADDWFC  MOVVBUF+B3              ; add 4th byte of MOVVBUF, save in

                        MOVFP32  A,MOVTMP               ; compute A/2
0478 7C9C               MOVFP   A+B0,MOVTMP+B0          ; move A(B0) to MOVTMP(B0)
0479 7D9D               MOVFP   A+B1,MOVTMP+B1          ; move A(B1) to MOVTMP(B1)
047A 7E9E               MOVFP   A+B2,MOVTMP+B2          ; move A(B2) to MOVTMP(B2)
047B 7F9F               MOVFP   A+B3,MOVTMP+B3          ; move A(B3) to MOVTMP(B3)

                        RRC32   MOVTMP
047C 1A1F               RLCF    MOVTMP+B3,W             ; move sign into carry bit
047D 191F               RRCF    MOVTMP+B3
047E 191E               RRCF    MOVTMP+B2
047F 191D               RRCF    MOVTMP+B1
0480 191C               RRCF    MOVTMP+B0

                        ADD32   MOVTMP,MOVPBUF          ; P(k)=P(k-1)+V(k-1)+A/2,
0481 6A1C               MOVFP   MOVTMP+B0,wreg  ; get lowest byte of MOVTMP into w
0482 0FA4               ADDWF   MOVPBUF+B0              ; add lowest byte of MOVPBUF, save in
0483 6A1D               MOVFP   MOVTMP+B1,wreg  ; get 2nd byte of MOVTMP into w
0484 11A5               ADDWFC  MOVPBUF+B1      ; add 2nd byte of MOVPBUF, save in
0485 6A1E               MOVFP   MOVTMP+B2,wreg  ; get 3rd byte of MOVTMP into w
0486 11A6               ADDWFC  MOVPBUF+B2      ; add 3rd byte of MOVPBUF, save in
0487 6A1F               MOVFP   MOVTMP+B3,wreg  ; get 4th byte of MOVTMP into w
0488 11A7               ADDWFC  MOVPBUF+B3      ; add 4th byte of MOVPBUF, save in )

0489 0002               return
```

```
;*****************************************************************

;*****************************************************************
; NAME: undoPosVel
;
; DESCRIPTION:  Backward iteration of the equations for trapezoidal
;               generation
;
;               V(k-1)=V(k)-A,          P(k-1)=P(k)-V(k-1)-A/2,
;
;               where abs(A)={AL,0} depending on the region of the
;               being executed. This routine is used to reverse a
;               to be made beyond a decision point.
;

                undoPosVel
                SUB32   A,MOVVBUF       ; V(k-1)=V(k)-A
048A 6A9C       MOVFP   A+B0,wreg       ; get lowest byte of A into w
048B 05A8       SUBWF   MOVVBUF+B0      ; sub lowest byte of MOVVBUF, save in
048C 6A9D       MOVFP   A+B1,wreg       ; get 2nd byte of A into w
048D 03A9       SUBWFB  MOVVBUF+B1      ; sub 2nd byte of MOVVBUF, save in
048E 6A9E       MOVFP   A+B2,wreg       ; get 3rd byte of A into w
048F 03AA       SUBWFB  MOVVBUF+B2      ; sub 3rd byte of MOVVBUF, save in
0490 6A9F       MOVFP   A+B3,wreg       ; get 4th byte of A into w
0491 03AB       SUBWFB  MOVVBUF+B3      ; sub 4th byte of MOVVBUF, save in

                SUB32   MOVVBUF,MOVPBUF ; P(k)-V(k-1)
0492 6AA8       MOVFP   MOVVBUF+B0,wreg ; get lowest byte of MOVVBUF into w
0493 05A4       SUBWF   MOVPBUF+B0      ; sub lowest byte of MOVPBUF, save in
0494 6AA9       MOVFP   MOVVBUF+B1,wreg ; get 2nd byte of MOVVBUF into w
0495 03A5       SUBWFB  MOVPBUF+B1      ; sub 2nd byte of MOVPBUF, save in
0496 6AAA       MOVFP   MOVVBUF+B2,wreg ; get 3rd byte of MOVVBUF into w
0497 03A6       SUBWFB  MOVPBUF+B2      ; sub 3rd byte of MOVPBUF, save in
0498 6AAB       MOVFP   MOVVBUF+B3,wreg ; get 4th byte of MOVVBUF into w
0499 03A7       SUBWFB  MOVPBUF+B3      ; sub 4th byte of MOVPBUF, save in

                MOVFP32  A,MOVTMP       ; compute A/2
049A 7C9C       MOVFP   A+B0,MOVTMP+B0  ; move A(B0) to MOVTMP(B0)
049B 7D9D       MOVFP   A+B1,MOVTMP+B1  ; move A(B1) to MOVTMP(B1)
049C 7E9E       MOVFP   A+B2,MOVTMP+B2  ; move A(B2) to MOVTMP(B2)
049D 7F9F       MOVFP   A+B3,MOVTMP+B3  ; move A(B3) to MOVTMP(B3)

                RRC32   MOVTMP
049E 1A1F       RLCF    MOVTMP+B3,W     ; move sign into carry bit
049F 191F       RRCF    MOVTMP+B3
04A0 191E       RRCF    MOVTMP+B2
04A1 191D       RRCF    MOVTMP+B1
04A2 191C       RRCF    MOVTMP+B0

                SUB32   MOVTMP,MOVPBUF  ; P(k-1)=P(k)-V(k-1)-A/2,
04A3 6A1C       MOVFP   MOVTMP+B0,wreg ; get lowest byte of MOVTMP into w
04A4 05A4       SUBWF   MOVPBUF+B0      ; sub lowest byte of MOVPBUF, save in
04A5 6A1D       MOVFP   MOVTMP+B1,wreg  ; get 2nd byte of MOVTMP into w
04A6 03A5       SUBWFB  MOVPBUF+B1      ; sub 2nd byte of MOVPBUF, save in
04A7 6A1E       MOVFP   MOVTMP+B2,wreg  ; get 3rd byte of MOVTMP into w
04A8 03A6       SUBWFB  MOVPBUF+B2      ; sub 3rd byte of MOVPBUF, save in
04A9 6A1F       MOVFP   MOVTMP+B3,wreg  ; get 4th byte of MOVTMP into w
04AA 03A7       SUBWFB  MOVPBUF+B3      ; sub 4th byte of MOVPBUF, save in

04AB 0002       return
```


© 1993 Microchip Technology Inc.

```
                ;*****************************************************************
                if _SERVO_PID
                include "pid.asm"          ; PID Algorithm
                ;*****************************************************************
                ;                          PID Servo Implementation
                ;
                ; Implement Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
                ;
                ;*****************************************************************

                ;*****************************************************************
                ; NAME:        doServo
                ;
                ; DESCRIPTION:  Performs the servo loop calculations.
                ;

                            doServo:

                            MOV16   POSERROR,U0             ; save new position error in
04AC 6A79                   MOVFP   POSERROR+B0,wreg        ; get byte of POSERROR into w
04AD 0184                   MOVWF   U0+B0                   ;move to U0(B0)
04AE 6A7A                   MOVFP   POSERROR+B1,wreg        ; get byte of POSERROR into w
04AF 0185                   MOVWF   U0+B1                   ; move to U0(B1)

                            LOADAB  U0,KP                   ; compute KP*U0
04B0 7C84                   MOVFP   U0+B0,AARG+B0           ; load lo byte of U0 to AARG
04B1 7D85                   MOVFP   U0+B1,AARG+B1           ; load hi byte of U0 to AARG
04B2 7E26                   MOVFP   KP+B0,BARG+B0           ; load lo byte of KP to BARG
04B3 7F27                   MOVFP   KP+B1,BARG+B1           ; load hi byte of KP to BARG
04B4 E12B                   call    Dmult
                            MOVPF32  DPX,Y                  ; Y=KP*U0
04B5 5880                   MOVPF   DPX+B0,Y+B0             ; move DPX(B0) to Y(B0)
04B6 5981                   MOVPF   DPX+B1,Y+B1             ; move DPX(B1) to Y(B1)
04B7 5A82                   MOVPF   DPX+B2,Y+B2             ; move DPX(B2) to Y(B2)
04B8 5B83                   MOVPF   DPX+B3,Y+B3             ; move DPX(B3) to Y(B3)
04B9 290A                   clrf    wreg                    ; if previous output saturated, do
04BA 3295                   cpfsgt  SATFLAG                 ; not accumulate integrator
04BB E552                   call    doIntegral

                            LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL
04BC 7C96                   MOVFP   INTEGRAL+B0,AARG+B0     ; load lo byte of INTEGRAL to AARG
04BD 7D97                   MOVFP   INTEGRAL+B1,AARG+B1     ; load hi byte of INTEGRAL to AARG
04BE 7E2A                   MOVFP   KI+B0,BARG+B0           ; load lo byte of KI to BARG
04BF 7F2B                   MOVFP   KI+B1,BARG+B1           ; load hi byte of KI to BARG
04C0 E12B                   call    Dmult

                            ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL
04C1 6A18                   MOVFP   DPX+B0,wreg             ; get lowest byte of DPX into w
04C2 0F80                   ADDWF   Y+B0                    ; add lowest byte of Y, save in Y(B0)
04C3 6A19                   MOVFP   DPX+B1,wreg             ; get 2nd byte of DPX into w
04C4 1181                   ADDWFC  Y+B1                    ; add 2nd byte of Y, save in Y(B1)
04C5 6A1A                   MOVFP   DPX+B2,wreg             ; get 3rd byte of DPX into w
04C6 1182                   ADDWFC  Y+B2                    ; add 3rd byte of Y, save in Y(B2)
04C7 6A1B                   MOVFP   DPX+B3,wreg             ; get 4th byte of DPX into w
04C8 1183                   ADDWFC  Y+B3                    ; add 4th byte of Y, save in Y(B3)

                            MOVFP16  U0,AARG                ; compute KV*(U0-U1)
04C9 7C84                   MOVFP   U0+B0,AARG+B0           ; move U0(B0) to AARG(B0)
04CA 7D85                   MOVFP   U0+B1,AARG+B1           ; move U0(B1) to AARG(B1)

                            SUB16   U1,AARG
04CB 6A86                   MOVFP   U1+B0,wreg              ; get lowest byte of U1 into w
04CC 051C                   SUBWF   AARG+B0                 ; sub lowest byte of AARG, save in
04CD 6A87                   MOVFP   U1+B1,wreg              ; get 2nd byte of U1 into w
04CE 031D                   SUBWFB  AARG+B1                 ; sub 2nd byte of AARG, save in
```

4

© 1993 Microchip Technology Inc.

```
                        MOVFP16 KV,BARG
04CF 7E28               MOVFP   KV+B0,BARG+B0          ; move KV(B0) to BARG(B0)
04D0 7F29               MOVFP   KV+B1,BARG+B1          ; move KV(B1) to BARG(B1)
04D1 E12B               call    Dmult
                        ADD32   DPX,Y          ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
04D2 6A18               MOVFP   DPX+B0,wreg    ; get lowest byte of DPX into w
04D3 0F80               ADDWF   Y+B0           ; add lowest byte of Y, save in Y(B0)
04D4 6A19               MOVFP   DPX+B1,wreg    ; get 2nd byte of DPX into w
04D5 1181               ADDWFC  Y+B1           ; add 2nd byte of Y, save in Y(B1)
04D6 6A1A               MOVFP   DPX+B2,wreg    ; get 3rd byte of DPX into w
04D7 1182               ADDWFC  Y+B2           ; add 3rd byte of Y, save in Y(B2)
04D8 6A1B               MOVFP   DPX+B3,wreg    ; get 4th byte of DPX into w
04D9 1183               ADDWFC  Y+B3           ; add 4th byte of Y, save in Y(B3)
                        MOV16   U0,U1          ; push errors into U(k-1)
04DA 6A84               MOVFP   U0+B0,wreg     ; get byte of U0 into w
04DB 0186               MOVWF   U1+B0          ; move to U1(B0)
04DC 6A85               MOVFP   U0+B1,wreg     ; get byte of U0 into w
04DD 0187               MOVWF   U1+B1          ; move to U1(B1)
04DE 290A               clrf    wreg
04DF 32B9               cpfsgt  SHIFTNUM
04E0 C4E9               goto    grabok
04E1 78B9               movfp   SHIFTNUM,TMP
        grabloop
                        RLC32   Y
04E2 8804               BCF     _carry
04E3 1B80               RLCF    Y+B0
04E4 1B81               RLCF    Y+B1
04E5 1B82               RLCF    Y+B2
04E6 1B83               RLCF    Y+B3
04E7 1718               decfsz  TMP
04E8 C4E2               goto    grabloop
        grabok
04E9 2995               clrf    SATFLAG
04EA 9F83               btfsc   Y+B3,MSB       ; saturate to middle 16 bits,
04EB C4F9               goto    negs           ; keeping top 10 bits for pw1dcH
        poss                                   ; and pw1dcL
04EC 6A82               movfp   Y+B2,wreg      ; check if Y >= 2**23
04ED B580               andlw   0x80
04EE 0983               iorwf   Y+B3
04EF 290A               clrf    wreg
04F0 3283               cpfsgt  Y+B3
04F1 C505               goto    zero6bits      ; if not, zero 6 bits
04F2 1595               incf    SATFLAG        ; if so, set Y=0x007FFFFF
04F3 2983               clrf    Y+B3           ; clear for debug purposes
04F4 B07F               movlw   0x7F
04F5 4A82               movpf   wreg,Y+B2
04F6 2B81               setf    Y+B1
04F7 2B80               setf    Y+B0
04F8 C505               goto    zero6bits
        negs
04F9 6A82               movfp   Y+B2,wreg      ; check if Y <= -2**23
04FA B37F               iorlw   0x7F
04FB 0B83               andwf   Y+B3
04FC 2B0A               setf    wreg
04FD 3083               cpfslt  Y+B3
04FE C505               goto    zero6bits      ; if not, zero 6 bits

04FF 2B95               setf    SATFLAG        ; if so, set Y = 0xFF800000
0500 2B83               setf    Y+B3
0501 2982               clrf    Y+B2
0502 8782               bsf     Y+B2,MSB
0503 2981               clrf    Y+B1
0504 2980               clrf    Y+B0
        zero6bits
                        MOV24   Y+B1,YPWM+B0   ; move Y to YPWM and zero 6 bits
0505 6A81               MOVFP   Y+B1+B0,wreg   ; get byte of Y+B1 into w
0506 4A88               MOVPF   wreg,YPWM+B0+B0 ; move to YPWM+B0(B0)
0507 6A82               MOVFP   Y+B1+B1,wreg   ; get byte of Y+B1 into w
0508 4A89               MOVPF   wreg,YPWM+B0+B1 ; move to YPWM+B0(B1)
```


© 1993 Microchip Technology Inc.

```
0509 6A83                    MOVFP   Y+B1+B2,wreg  ; get byte of Y+B1 into w
050A 4A8A                    MOVPF   wreg,YPWM+B0+B2 ; move to YPWM+B0(B2)
                 doTorque            ; entry point for torque mode
050B B0C0                    movlw   0xC0
050C 0B88                    andwf   YPWM+B0
050D 9F89                    btfsc   YPWM+B1,MSB
050E C516                    goto    tmlimit
                 tplimit
050F 9692                    btfss   EXTSTAT,bit6
0510 C51C                    goto    mplimitok
                             CLR32   YPWM
0511 2988                    CLRF    YPWM+B0
0512 2989                    CLRF    YPWM+B1
0513 298A                    CLRF    YPWM+B2
0514 298B                    CLRF    YPWM+B3
0515 C51C                    goto    mplimitok
                 tmlimit
0516 9592                    btfss   EXTSTAT,bit5
0517 C51C                    goto    mplimitok
                             CLR32   YPWM
0518 2988                    CLRF    YPWM+B0
0519 2989                    CLRF    YPWM+B1
051A 298A                    CLRF    YPWM+B2
051B 298B                    CLRF    YPWM+B3
                 mplimitok
051C B07F                    movlw   PW1DCH_INIT     ; adjustment from bipolar to unipolar
051D 4A19                    movpf   wreg,TMP+B1     ; for 50% duty cycle
051E B0C0                    movlw   PW1DCL_INIT
051F 4A18                    movpf   wreg,TMP+B0
                             ADD16   TMP,YPWM
0520 6A18                    MOVFP   TMP+B0,wreg     ; get lowest byte of TMP into w
0521 0F88                    ADDWF   YPWM+B0         ; add lowest byte of YPWM, save in YPWM(B0)
0522 6A19                    MOVFP   TMP+B1,wreg     ; get 2nd byte of TMP into w
0523 1189                    ADDWFC  YPWM+B1         ; add 2nd byte of YPWM, save in YPWM(B1)
0524 2919                    clrf    TMP+B1          ; correct by 1 LSB
0525 B040                    movlw   0x40            ; add one to bit5 of pw1dcL
0526 4A18                    movpf   wreg,TMP+B0
                             ADD16   TMP,YPWM
0527 6A18                    MOVFP   TMP+B0,wreg     ; get lowest byte of TMP into w
0528 0F88                    ADDWF   YPWM+B0         ; add lowest byte of YPWM, save in YPWM(B0)
0529 6A19                    MOVFP   TMP+B1,wreg     ; get 2nd byte of TMP into w
052A 1189                    ADDWFC  YPWM+B1         ; add 2nd byte of YPWM, save in YPWM(B1)

                 testmax
052B 291A                    clrf    TMP+B2          ; check pwm maximum limit
052C 298A                    clrf    YPWM+B2         ; LMD18200 must have a minimum pulse
052D 298B                    clrf    YPWM+B3         ; so duty cycle must not be 0 or 100%
                             MOVFP16  YPWMAX,TMP
052E 788E                    MOVFP   YPWMAX+B0,TMP+B0; move YPWMAX(B0) to TMP(B0)
052F 798F                    MOVFP   YPWMAX+B1,TMP+B1; move YPWMAX(B1) to TMP(B1)
                             SUB24   YPWM,TMP
0530 6A88                    MOVFP   YPWM+B0,wreg    ; get lowest byte of YPWM into w
0531 0518                    SUBWF   TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0532 6A89                    MOVFP   YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0533 0319                    SUBWFB  TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0534 6A8A                    MOVFP   YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0535 031A                    SUBWFB  TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)
0536 971A                    btfss   TMP+B2,MSB
0537 C53D                    goto    testmin
                             MOV16   YPWMAX,YPWM     ; saturate to max
0538 6A8E                    MOVFP   YPWMAX+B0,wreg  ; get byte of YPWMAX into w
0539 0188                    MOVWF   YPWM+B0         ; move to YPWM(B0)
053A 6A8F                    MOVFP   YPWMAX+B1,wreg  ; get byte of YPWMAX into w
053B 0189                    MOVWF   YPWM+B1         ; move to YPWM(B1)
053C C54E                    goto    limitok
                 testmin
053D 291A                    clrf    TMP+B2          ;check pwm minimum limit
053E 298A                    clrf    YPWM+B2
053F 298B                    clrf    YPWM+B3
```


© 1993 Microchip Technology Inc.

```
                        MOVFP16  YPWMIN,TMP
0540 788C               MOVFP    YPWMIN+B0,TMP+B0 ; move YPWMIN(B0) to TMP(B0)
0541 798D               MOVFP    YPWMIN+B1,TMP+B1 ; move YPWMIN(B1) to TMP(B1)
                        SUB24    YPWM,TMP
0542 6A88               MOVFP    YPWM+B0,wreg    ; get lowest byte of YPWM into w
0543 0518               SUBWF    TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0544 6A89               MOVFP    YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0545 0319               SUBWFB   TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0546 6A8A               MOVFP    YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0547 031A               SUBWFB   TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)
0548 9F1A               btfsc    TMP+B2,MSB
0549 C54E               goto     limitok
                        MOV16    YPWMIN,YPWM     ; saturate to min
054A 6A8C               MOVFP    YPWMIN+B0,wreg  ; get byte of YPWMIN into w
054B 0188               MOVWF    YPWM+B0         ; move to YPWM(B0)
054C 6A8D               MOVFP    YPWMIN+B1,wreg  ; get byte of YPWMIN into w
054D 0189               MOVWF    YPWM+B1         ; move to YPWM(B1)
                limitok
054E B803               movlb    bank3           ; set new duty cycle
054F 7088               movfp    YPWM+B0,pw1dcl
0550 7289               movfp    YPWM+B1,pw1dch

0551 0002               return
                ;*****************************************************************

                ;*****************************************************************
                ; NAME:         doIntegral
                ;
                ; DESCRIPTION:  Evaluates the integral for the servo calculations.
                ;
                        doIntegral
                        ADD16    U0,INTEGRAL     ; do integral
0552 6A84               MOVFP    U0+B0,wreg      ; get lowest byte of U0 into w
0553 0F96               ADDWF    INTEGRAL+B0     ; add lowest byte of INTEGRAL, save in
0554 6A85               MOVFP    U0+B1,wreg      ; get 2nd byte of U0 into w
0555 1197               ADDWFC   INTEGRAL+B1     ; add 2nd byte of INTEGRAL, save in

0556 0002               return
                ;*****************************************************************

                endif

                if _SERIAL_IO
                include "serial.asm"                    ; Serial I/O Routines
                ;*****************************************************************
                ;
                ;                       Serial I/O & Utility Functions
                ;
                ;*****************************************************************

                ;*****************************************************************
                ; NAME:         IdleFunction
                ; DESCRIPTION:  This routine will perform work while doing waits in
                ;               serial I/O functions.
                ;
                IdleFunction
0557 0004       CLRWDT
0558 0002       return

                ;*****************************************************************

                ;*****************************************************************
                ; NAME:         DoCommand
```


© 1993 Microchip Technology Inc.

```
                        ;
                        ; DESCRIPTION:  Search command table for command and execute it.
                        ;

                        DoCommand

0559 B059                       movlw   (CMD_TABLE & 0xff)      ; CMD_TABLE LSB
055A 4A0D                       movpf   wreg,tblptrl
055B B007                       movlw   page CMD_TABLE          ; CMD_TABLE MSB
055C 4A0E                       movpf   wreg,tblptrh

055D AB3A                       tablrd  1,1,CMDTEMP
                        tryNextCmd

055E A93A                       tablrd  0,1,CMDTEMP             ; read entry from table
055F A23B                       tlrd    1,CMDPTRH
0560 A93C                       tablrd  0,1,CMDPTRL

0561 6A3A                       movfp   CMDTEMP,wreg
0562 30C1                       cpfslt  ZERO

0563 C56E                       goto    noCommand               ; error if end of table

0564 3139                       cpfseq  CMDCHAR
0565 C55E                       goto    tryNextCmd

0566 E679                       call    PutChar                 ; echo command

0567 633B                       movfp   CMDPTRH,pclath          ; indirect jump to command routine
0568 623C                       movfp   CMDPTRL,pcl
0569 0000                       NOP
                        cmdFinish
056A E679                       call    PutChar                 ; send response character from
                                                                ; command routine followed by CR
056B B00D                       movlw   CR
056C E679                       call    PutChar

056D C124                       goto    PollingLoop

                         noCommand
056E B03F                       movlw   CMD_BAD                  ; send error character
056F C56A                       goto    cmdFinish

                        ;*************************************************************************

                        ;*************************************************************************
                        ; NAME:         do_null
                        ;
                        ; DESCRIPTION:  The do nothing command used to determine if the chip is
                        ;               working. Initiated by a carriage return.

                        do_null
0570 B021                       movlw   CMD_OK
0571 C56A                       goto    cmdFinish

                        ;*************************************************************************

                        ;*************************************************************************
                        ; NAME:         do_move
                        ; DESCRIPTION:  Commands the axis to move to a new position or velocity.
                        ;               Position data is relative, and in encoder counts. Velocity
                        ;               data is absolute, and in encoder counts/sample time multi-
                        ;               plied by 256. All moves are performed by the controller such
                        ;               that velocity and acceleration limits set into parameter
                        ;               memory will not be violated. All move commands are kept in a
                        ;               one deep FIFO buffer. The command in the buffer is executed
                        ;               as soon as the currently executed command is complete.
                        ;
```

© 1993 Microchip Technology Inc.

```
                        ;
                        ; ARGUMENTS:    M [800000,7FFFFF]
                        ;
                        do_move

                                if      DECIO
0572 E6CC                       call    GetDecVal
                                else

                                call    GetVal

                                endif
0573 9F93                       btfsc   MOVSTAT,bit7        ; test if buffer available
0574 C57E                       goto    bufoverflow
                                MOV24   VALBUF,NMOVVAL      ; if so, accept value into NMOVVAL
0575 6A31                       MOVFP   VALBUF+B0,wreg      ; get byte of VALBUF into w
0576 4A5B                       MOVPF   wreg,NMOVVAL+B0     ; move to NMOVVAL(B0)
0577 6A32                       MOVFP   VALBUF+B1,wreg      ; get byte of VALBUF into w
0578 4A5C                       MOVPF   wreg,NMOVVAL+B1     ; move to NMOVVAL(B1)
0579 6A33                       MOVFP   VALBUF+B2,wreg      ; get byte of VALBUF into w
057A 4A5D                       MOVPF   wreg,NMOVVAL+B2     ; move to NMOVVAL(B2)
057B 8793                       bsf     MOVSTAT,bit7        ; set buffer full flag
057C B021                       movlw   CMD_OK
057D C56A                       goto    cmdFinish

                        bufoverflow
057E B03F                       movlw   CMD_BAD ; else, return error
057F C56A                       goto    cmdFinish

                        ;*****************************************************************************
                        ;*****************************************************************************
                        ; NAME:         do_mode
                        ;
                        ; DESCRIPTION:  An argument of "P" will cause all subsequent move commands
                        ;               to be incremental position moves. A "V" argument will cause
                        ;               all subsequent moves to be absolute velocity moves.
                        ;
                        ; ARGUMENTS:    O [P,V]
                        ;

                        do_mode

0580 E557                       call    IdleFunction    ; get single character loop
0581 E681                       call    GetChk
0582 31C2                       cpfseq  ONE
0583 C580                       goto    do_mode
0584 E676                       call    GetChar
0585 4A4D                       movpf   wreg,STRVALL
0586 2991                       clrf    MODETYPE        ; MODETYPE=0 for position moves
                        testP
0587 B050                       movlw   'P'             ; position moves for type P
0588 314D                       cpfseq  STRVALL
0589 C58B                       goto    testV
058A C598                       goto    modeok
                        testV
058B B056                       movlw   'V'             ; velocity moves for type V
058C 314D                       cpfseq  STRVALL
058D C590                       goto    testT
058E 1591                       incf    MODETYPE        ; MODETYPE=1 for velocity moves
058F C598                       goto    modeok
                        testT
0590 B054                       movlw   'T'             ; TORQUE Moves for type 'T'
0591 314D                       cpfseq  STRVALL
0592 C596                       goto    modeerror
0593 2B91                       setf    MODETYPE        ; MODETYPE=-1 for torque moves
```


© 1993 Microchip Technology Inc.

```
0594 2990                clrf    SERVOFLAG       ; disable servo
0595 C598                goto    modeok
                 modeerror
0596 B03F                movlw   CMD_BAD         ; mode error
0597 C56A                goto    cmdFinish
                 modeok
0598 6A4D                movfp   STRVALL,wreg    ; echo type character
0599 E679                call    PutChar

059A B021                movlw   CMD_OK
059B C56A                goto    cmdFinish

                 ;*****************************************************************************

                 ;*****************************************************************************
                 ; NAME:         do_setparameter
                 ;
                 ; DESCRIPTION:  Sets controller parameters to the value given.
                 ;
                 ;       Parameter               #               Range
                 ;
                 ;       VL=velocity limit       0               [0,7FFFFF]
                 ;       AL=acceleration limit   1               [0,7FFFFF]
                 ;
                 ;       KP=proportional gain    2               [8000,7FFF]
                 ;       KP=velocity gain        3               [8000,7FFF]
                 ;       KP=integral gain        4               [8000,7FFF]
                 ;
                 ;       IM=integrator mode      5               [0,3]
                 ;
                 ;       FV=velocity FF          6               [8000,7FFF] : Not Imple
                 ;       FA=acceleration FF      7               [8000,7FFF] : Not Imple
                 ;
                 ;
                 ; ARGUMENTS:    S [0,FF] [800000,7FFFFF]
                 ;

                 do_setparameter

059C E669                call    GetPar                  ; get parameter number

059D B008                movlw   NUMPAR                  ; check if in range [0,NUMPAR]
059E 3031                cpfslt  VALBUF+B0
059F C5C1                goto    Serror

05A0 B07C                movlw   (PAR_TABLE & 0xff)      ; PAR_TABLE LSB
05A1 4A0D                movpf   wreg,tblptrl
05A2 B007                movlw   page PAR_TABLE          ; PAR_TABLE MSB
05A3 4A0E                movpf   wreg,tblptrh
05A4 AB3D                tablrd  1,1,PARTEMP

                 setNextPar

05A5 A23D                tlrd    1,PARTEMP               ; read entry from table
05A6 A93E                tablrd  0,1,PARLEN
05A7 A93F                tablrd  0,1,PARPTR

05A8 B008                movlw   NUMPAR                  ; error if end of table
05A9 303D                cpfslt  PARTEMP
05AA C5C1                goto    Serror

05AB 6A3D                movfp   PARTEMP,wreg
05AC 3131                cpfseq  VALBUF+B0
05AD C5A5                goto    setNextPar

05AE 6A3F                movfp   PARPTR,wreg             ; pointer to parameter in fsr1
05AF 690A                movfp   wreg,fsr1
```

4

© 1993 Microchip Technology Inc.

```
                        if      DECIO                   ; get new value in VALBUF
05B0 E6CC               call    GetDecVal
                        else
                        call    GetVal
                        endif
05B1 B031               movlw   VALBUF                  ; pointer to VALBUF in fsr0
05B2 610A               movfp   wreg,fsr0
                         AUTOINC                        ; set autoincrement
05B3 8404               BSF     _fs0
05B4 8D04               BCF     _fs1
05B5 8604               BSF     _fs2
05B6 8F04               BCF     _fs3

                    setGetMore
05B7 6800               movfp   indf0,indf1             ; move new value to parameter
05B8 073E               decf    PARLEN
05B9 333E               tstfsz  PARLEN
05BA C5B7               goto    setGetMore
                         AUTONO                         ; no autoincrement
05BB 8404               BSF     _fs0
05BC 8504               BSF     _fs1
05BD 8604               BSF     _fs2
05BE 8704               BSF     _fs3

05BF B021               movlw   CMD_OK
05C0 C56A               goto    cmdFinish

                    Serror
05C1 B03F               movlw   CMD_BAD
05C2 C56A               goto    cmdFinish
                    ;*****************************************************************
                    ;*****************************************************************
                    ; NAME:         do_readparameter
                    ;
                    ; DESCRIPTION:  Returns the present value of a parameter.
                    ;
                    ; ARGUMENTS:    R [0,FF]
                    ;
                    ; RETURNS:      The present value of the requested parameter is returned.

                    do_readparameter
05C3 E669               call    GetPar                  ; get parameter number
05C4 B008               movlw   NUMPAR                  ; check if in range [0,NUMPAR]
05C5 3031               cpfslt  VALBUF+B0
05C6 C5EB               goto    Rerror
05C7 B07C               movlw   (PAR_TABLE & 0xff)      ; PAR_TABLE LSB
05C8 4A0D               movpf   wreg,tblptrl
05C9 B007               movlw   page PAR_TABLE          ; PAR_TABLE MSB
05CA 4A0E               movpf   wreg,tblptrh
05CB AB3D               tablrd  1,1,PARTEMP

                    readNextPar
05CC A23D               tlrd    1,PARTEMP               ; read entry from table
```


© 1993 Microchip Technology Inc.

```
05CD A93E                       tablrd  0,1,PARLEN
05CE A93F                       tablrd  0,1,PARPTR

05CF B008                       movlw   NUMPAR                  ; error if end of table
05D0 303D                       cpfslt  PARTEMP
05D1 C5EB                       goto    Rerror

05D2 6A3D                       movfp   PARTEMP,wreg
05D3 3131                       cpfseq  VALBUF+B0
05D4 C5CC                       goto    readNextPar

05D5 6A3F                       movfp   PARPTR,wreg             ; pointer to parameter in fsr1
05D6 690A                       movfp   wreg,fsr1
05D7 B031                       movlw   VALBUF                  ; pointer to VALBUF in fsr1
05D8 610A                       movfp   wreg,fsr0
                                 AUTOINC                        ; set autoincrement

05D9 8404                       BSF     _fs0
05DA 8D04                       BCF     _fs1
05DB 8604                       BSF     _fs2
05DC 8F04                       BCF     _fs3

                                CLR24   VALBUF                  ; clear old VALBUF
05DD 2931                       CLRF    VALBUF+B0
05DE 2932                       CLRF    VALBUF+B1
05DF 2933                       CLRF    VALBUF+B2

                    readGetMore
05E0 6008                       movfp   indf1,indf0     ; read parameter into VALBUF
05E1 073E                       decf    PARLEN
05E2 333E                       tstfsz  PARLEN
05E3 C5E0                       goto    readGetMore
                   AUTONO                               ; no autoincrement

05E4 8404                       BSF     _fs0
05E5 8504                       BSF     _fs1
05E6 8604                       BSF     _fs2
05E7 8704                       BSF     _fs3

                                if      DECIO           ; send parameter value
05E8 E728                       call    PutDecVal
                                else

                                call    PutVal

                                endif
05E9 B021                       movlw   CMD_OK
05EA C56A                       goto    cmdFinish

                    Rerror
05EB B03F                       movlw   CMD_BAD
05EC C56A                       goto    cmdFinish

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME: do_shutter
                    ;
                    ; DESCRIPTION:  Returns the time (in sample time counts [0,FFFF]) since the
                    ;               start of the present move and captures the commanded and
                    ;               measured values of position and velocity at the time of the
                    ;               command.
```

© 1993 Microchip Technology Inc.

```
                        ;
                        ;
                        ; ARGUMENTS:    C
                        ;
                        ; RETURNS:      The time since the start of the present move is returned.
                        ;

                        do_shutter

                                MOV24   POSITION,CPOSITION      ; capture commanded position

05ED 6A55               MOVFP   POSITION+B0,wreg        ; get byte of POSITION into w
05EE 4A40               MOVPF   wreg,CPOSITION+B0       ; move to CPOSITION(B0)
05EF 6A56               MOVFP   POSITION+B1,wreg        ; get byte of POSITION into w
05F0 4A41               MOVPF   wreg,CPOSITION+B1       ; move to CPOSITION(B1)
05F1 6A57               MOVFP   POSITION+B2,wreg        ; get byte of POSITION into w
05F2 4A42               MOVPF   wreg,CPOSITION+B2       ; move to CPOSITION(B2)

                        MOV24   VELOCITY,CVELOCITY      ; capture commanded velocity

05F3 6A58               MOVFP   VELOCITY+B0,wreg        ; get byte of VELOCITY into w
05F4 4A43               MOVPF   wreg,CVELOCITY+B0       ; move to CVELOCITY(B0)
05F5 6A59               MOVFP   VELOCITY+B1,wreg        ; get byte of VELOCITY into w
05F6 4A44               MOVPF   wreg,CVELOCITY+B1       ; move to CVELOCITY(B1)
05F7 6A5A               MOVFP   VELOCITY+B2,wreg        ; get byte of VELOCITY into w
05F8 4A45               MOVPF   wreg,CVELOCITY+B2       ; move to CVELOCITY(B2)

                        MOV24   MPOSITION,CMPOSITION    ; capture measured position

05F9 6A72               MOVFP   MPOSITION+B0,wreg       ; get byte of MPOSITION into w
05FA 4A46               MOVPF   wreg,CMPOSITION+B0      ; move to CMPOSITION(B0)
05FB 6A73               MOVFP   MPOSITION+B1,wreg       ; get byte of MPOSITION into w
05FC 4A47               MOVPF   wreg,CMPOSITION+B1      ; move to CMPOSITION(B1)
05FD 6A74               MOVFP   MPOSITION+B2,wreg       ; get byte of MPOSITION into w
05FE 4A48               MOVPF   wreg,CMPOSITION+B2      ; move to CMPOSITION(B2)

                        MOV24   MVELOCITY,CMVELOCITY    ; capture measured velocity

05FF 6A75               MOVFP   MVELOCITY+B0,wreg       ; get byte of MVELOCITY into w
0600 4A49               MOVPF   wreg,CMVELOCITY+B0      ; move to CMVELOCITY(B0)
0601 6A76               MOVFP   MVELOCITY+B1,wreg       ; get byte of MVELOCITY into w
0602 4A4A               MOVPF   wreg,CMVELOCITY+B1      ; move to CMVELOCITY(B1)
0603 6A77               MOVFP   MVELOCITY+B2,wreg       ; get byte of MVELOCITY into w
0604 4A4B               MOVPF   wreg,CMVELOCITY+B2      ; move to CMVELOCITY(B2)

0605 2933               clrf    VALBUF+B2
                        MOV16   MOVTIME,VALBUF          ; capture move time, move to VALBUF

0606 6A67               MOVFP   MOVTIME+B0,wreg         ; get byte of MOVTIME into w
0607 0131               MOVWF   VALBUF+B0               ; move to VALBUF(B0)
0608 6A68               MOVFP   MOVTIME+B1,wreg         ; get byte of MOVTIME into w
0609 0132               MOVWF   VALBUF+B1               ; move to VALBUF(B1)

                        if      DECIO

060A E728               call    PutDecVal

                        else

                        call    PutVal

                        endif

060B B021               movlw   CMD_OK
060C C56A               goto    cmdFinish
```


© 1993 Microchip Technology Inc.

```
                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME: do_readcomposition
                    ;
                    ; DESCRIPTION:  Returns the commanded position count which was captured
                    ;               during the last shutter command.
                    ;
                    ; ARGUMENTS:    P
                    ;
                    ; RETURNS:      The last captured position count is returned. [800000,7FFFFF]
                    ;
                    do_readcomposition

                            MOV24   CPOSITION,VALBUF        ; move CPOSITION to VALBUF
060D 6A40                   MOVFP   CPOSITION+B0,wreg       ; get byte of CPOSITION into w
060E 4A31                   MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
060F 6A41                   MOVFP   CPOSITION+B1,wreg       ; get byte of CPOSITION into w
0610 4A32                   MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
0611 6A42                   MOVFP   CPOSITION+B2,wreg       ; get byte of CPOSITION into w
0612 4A33                   MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)

                            if      DECIO
0613 E728                   call    PutDecVal
                            else
                            call    PutVal
                            endif
0614 B021                   movlw   CMD_OK
0615 C56A                   goto    cmdFinish

                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME: do_readcomvelocity
                    ;
                    ; DESCRIPTION:  Returns the commanded velocity multiplied by 256 which was
                    ;               captured during the last shutter command.
                    ;
                    ; ARGUMENTS:    V
                    ;
                    ; RETURNS:      The last captured commanded velocity times 256 is returned.
                    ;               [800000,7FFFFF]
                    ;
                    do_readcomvelocity

                            MOV24   CVELOCITY,VALBUF        ; move commanded velocity to VALBUF
0616 6A43                   MOVFP   CVELOCITY+B0,wreg       ; get byte of CVELOCITY into w
0617 4A31                   MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
0618 6A44                   MOVFP   CVELOCITY+B1,wreg       ; get byte of CVELOCITY into w
0619 4A32                   MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
061A 6A45                   MOVFP   CVELOCITY+B2,wreg       ; get byte of CVELOCITY into w
061B 4A33                   MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)

                            if      DECIO
061C E728                   call    PutDecVal
                            else
                            call    PutVal
```


© 1993 Microchip Technology Inc.

```
                        endif
061D B021               movlw   CMD_OK
061E C56A               goto    cmdFinish
                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME: do_readactposition
                ;
                ; DESCRIPTION:  Returns the measured position count which was captured
                ;               during the last shutter command.
                ;
                ; ARGUMENTS:    p
                ;
                ; RETURNS:      The last captured measured position count is returned.
                ;               [800000,7FFFFF]
                ;
                do_readactposition
                MOV24   CMPOSITION,VALBUF               ; move measured position to

061F 6A46               MOVFP   CMPOSITION+B0,wreg      ; get byte of CMPOSITION into w
0620 4A31               MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
0621 6A47               MOVFP   CMPOSITION+B1,wreg      ; get byte of CMPOSITION into w
0622 4A32               MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
0623 6A48               MOVFP   CMPOSITION+B2,wreg      ; get byte of CMPOSITION into w
0624 4A33               MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)

                        if      DECIO

0625 E728               call    PutDecVal

                        else

                        call    PutVal

                        endif

0626 B021               movlw   CMD_OK
0627 C56A               goto    cmdFinish
                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME: do_readactvelocity
                ;
                ; DESCRIPTION:  Returns the measured velocity multiplied by 256 which was
                ;               captured during the last shutter command.
                ;
                ; ARGUMENTS:    v
                ;
                ; RETURNS:      The last captured measured velocity times 256 is returned.
                ;               [800000,7FFFFF]
                ;
                do_readactvelocity
                MOV24   CMVELOCITY,VALBUF               ; move measured velocity to

0628 6A49               MOVFP   CMVELOCITY+B0,wreg      ; get byte of CMVELOCITY into w
0629 4A31               MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
062A 6A4A               MOVFP   CMVELOCITY+B1,wreg      ; get byte of CMVELOCITY into w
062B 4A32               MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
062C 6A4B               MOVFP   CMVELOCITY+B2,wreg      ; get byte of CMVELOCITY into w
062D 4A33               MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)
```


© 1993 Microchip Technology Inc.

```
                        if      DECIO

062E E728               call    PutDecVal

                        else
                        call    PutVal

                        endif

062F B021               movlw   CMD_OK
0630 C56A               goto    cmdFinish

                ;*****************************************************************************
                ;*****************************************************************************
                ; NAME: do_externalstatus
                ;
                ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                ;               the bits in the external status register. Issuing this
                ;               command will clear all the bits in the external status
                ;               register unless the event which set the bit is still true.
                ;
                ;
                ; ARGUMENTS:    X
                ;
                ; RETURNS:      The external status register is returned.
                ;

                do_externalstatus

0631 8406               bsf     _glintd
0632 6A92               movfp   EXTSTAT,wreg
0633 2992               clrf    EXTSTAT
0634 8C06               bcf     _glintd
0635 E688               call    PutHex

0636 B021               movlw   CMD_OK
0637 C56A               goto    cmdFinish

                ;*****************************************************************************
                ;*****************************************************************************
                ; NAME: do_movestatus
                ;
                ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                ;               the bits in the move status register. Issuing this command
                ;               will clear all the bits in the move status register unless
                ;               the event which set the bit is still true.
                ;
                ; ARGUMENTS:    Y
                ;
                ; RETURNS:      The move status register is returned.
                ;

                do_movestatus

0638 6A93               movfp   MOVSTAT,wreg
0639 E688               call    PutHex

063A B021               movlw   CMD_OK
063B C56A               goto    cmdFinish
```

© 1993 Microchip Technology Inc.

```
                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME:         do_readindposition
                    ;
                    ; DESCRIPTION:  Returns the last index position captured in position counts.
                    ;
                    ; ARGUMENTS:    I
                    ;
                    ; RETURNS:      The last captured index position is returned.
                    ;
                    do_readindposition

                            MOV24   INDEXPOS,VALBUF         ; move measured velocity to VALBUF

063C 6AB6                   MOVFP   INDEXPOS+B0,wreg        ; get byte of INDEXPOS into w
063D 4A31                   MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
063E 6AB7                   MOVFP   INDEXPOS+B1,wreg        ; get byte of INDEXPOS into w
063F 4A32                   MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
0640 6AB8                   MOVFP   INDEXPOS+B2,wreg        ; get byte of INDEXPOS into w
0641 4A33                   MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)

                            if      DECIO

0642 E728                   call    PutDecVal

                            else

                            call    PutVal

                            endif

0643 B021                   movlw   CMD_OK
0644 C56A                   goto    cmdFinish

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME:         do_setposition
                    ;
                    ; DESCRIPTION:  Sets the measured and commanded position to the value given.
                    ;               This command should not be sent unless the move FIFO buffer
                    ;
                    ; ARGUMENTS:    H [800000,7FFFFF]
                    ;
                    do_setposition

                            if      DECIO

0645 E6CC                   call    GetDecVal

                            else

                            call    GetVal

                            endif

                            MOV24   VALBUF,POSITION

0646 6A31                   MOVFP   VALBUF+B0,wreg          ; get byte of VALBUF into w
0647 4A55                   MOVPF   wreg,POSITION+B0        ; move to POSITION(B0)
0648 6A32                   MOVFP   VALBUF+B1,wreg          ; get byte of VALBUF into w
0649 4A56                   MOVPF   wreg,POSITION+B1        ; move to POSITION(B1)
064A 6A33                   MOVFP   VALBUF+B2,wreg          ; get byte of VALBUF into w
064B 4A57                   MOVPF   wreg,POSITION+B2        ; move to POSITION(B2)
```


© 1993 Microchip Technology Inc.

```
                        MOV24   VALBUF,MPOSITION

064C 6A31               MOVFP   VALBUF+B0,wreg          ; get byte of VALBUF into w
064D 4A72               MOVPF   wreg,MPOSITION+B0       ; move to MPOSITION(B0)
064E 6A32               MOVFP   VALBUF+B1,wreg          ; get byte of VALBUF into w
064F 4A73               MOVPF   wreg,MPOSITION+B1       ; move to MPOSITION(B1)
0650 6A33               MOVFP   VALBUF+B2,wreg          ; get byte of VALBUF into w
0651 4A74               MOVPF   wreg,MPOSITION+B2       ; move to MPOSITION(B2)

                        CLR32   Y

0652 2980               CLRF    Y+B0
0653 2981               CLRF    Y+B1
0654 2982               CLRF    Y+B2
0655 2983               CLRF    Y+B3

0656 B021               movlw   CMD_OK
0657 C56A               goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME:         do_reset
                ;
                ; DESCRIPTION:  Performs a software reset.
                ;
                ; ARGUMENTS:    Z
                ;
                do_reset

0658 B021               movlw   CMD_OK
0659 E679               call    PutChar
065A C021               goto    Startup


                ;*****************************************************************************
                ; NAME:         do_stop
                ;
                ; DESCRIPTION:  Stops servo by clearing SERVOFLAG.
                ;
                do_stop

065B 2990               clrf    SERVOFLAG

065C B021               movlw   CMD_OK
065D C56A               goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME:         do_capture
                ;
                do_capture

                  if     ((_PICMASTER_DEBUG == 1) && (DECIO == 1))

065E E6CC               call    GetDecVal

                  endif

                  if     ((_PICMASTER_DEBUG == 1) && (DECIO == 0))

                  call    GetVal

                  endif
                  if    _PICMASTER_DEBUG == 1
```

```
                        MOV16   VALBUF,CAPCOUNT
065F 6A31               MOVFP   VALBUF+B0,wreg  ; get byte of VALBUF into w
0660 01BC               MOVWF   CAPCOUNT+B0     ; move to CAPCOUNT(B0)
0661 6A32               MOVFP   VALBUF+B1,wreg  ; get byte of VALBUF into w
0662 01BD               MOVWF   CAPCOUNT+B1     ; move to CAPCOUNT(B1)

                        MOV16   VALBUF,CAPTMP

0663 6A31               MOVFP   VALBUF+B0,wreg  ; get byte of VALBUF into w
0664 01BE               MOVWF   CAPTMP+B0       ; move to CAPTMP(B0)
0665 6A32               MOVFP   VALBUF+B1,wreg  ; get byte of VALBUF into w
0666 01BF               MOVWF   CAPTMP+B1       ; move to CAPTMP(B1)

0667 B021               movlw   CMD_OK
0668 C56A               goto    cmdFinish
                    endif

                    ;*************************************************************************
                    ; NAME:         GetPar
                    ;
                    ; DESCRIPTION:  Get a parameter number [0,FF] from the serial port and place
                    ;               it in VALBUF+B0.
                    ;

                    GetPar

                     CLR24   VALBUF

0669 2931               CLRF    VALBUF+B0
066A 2932               CLRF    VALBUF+B1
066B 2933               CLRF    VALBUF+B2

066C E6B2               call    GetHex
066D 6A4E               movfp   HEXVAL,wreg
066E B50F               andlw   0x0F
066F 4A31               movpf   wreg,VALBUF+B0
0670 1D31               swapf   VALBUF+B0

0671 E6B2               call    GetHex
0672 6A31               movfp   VALBUF+B0,wreg

0673 0E4E               addwf   HEXVAL,W
0674 4A31               movpf   wreg,VALBUF+B0

0675 0002               return

                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:         GetChar
                    ;
                    ; DESCRIPTION:  Get character from receive buffer.
                    ;
                    GetChar

0676 B800               movlb   bank0           ; set bank0
0677 540A               movpf   rcreg,wreg      ; receive character

0678 0002               return

                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:         PutChar
                    ;
                    ; DESCRIPTION:  send character out the serial port
                    ;
```


© 1993 Microchip Technology Inc.

```
                        ; ARGUMENTS:    wreg contains byte to be transmitted
                        ;
                        PutChar
0679 B801                       movlb   bank1           ; set bank1
                        bufwait                         ; is transmit buffer empty?
067A 9116                       btfss   _tbmt
067B C67A                       goto    bufwait

067C B800                       movlb   bank0           ; set bank0
                        shfwait
067D 9115                       btfss   _trmt           ; is transmit shift register empty?
067E C67D                       goto    shfwait

067F 4A16                       movpf   wreg,txreg      ; if so, send character

0680 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         GetChk
                        ;
                        ; DESCRIPTION:  Check if character is in receive buffer.
                        ;
                        GetChk
0681 B801                       movlb   bank1           ; set bank1
0682 560A                       movpf   pir,wreg
0683 B501                       andlw   CHARREADY       ; return status in wreg
0684 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME: PutDec
                        ;
                        ; DESCRIPTION:  Converts a hex value [0,F] in wreg to its ASCII equivalent.
                        ;               The upper nibble of wreg is assumed to be zero.
                        ;
                        ; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII decimal
                        ;

                                if      DECIO

                        PutDec
0685 B130                       addlw   0x30            ; convert to ASCII
0686 E679                       call    PutChar
0687 0002                       return

                                endif

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME: PutHex
                        ;
                        ; DESCRIPTION:  Convert the wreg value to ASCII hexadecimal. The output
                        ;               format is two digits with the A-F parts in upper case and
                        ;               leading zeros. The result is sent out the serial port with
                        ;               PutChar.
                        ;
                        ; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII hex
                        ;

                        PutHex
```


© 1993 Microchip Technology Inc.

```
0688 4A4E                   movpf   wreg,HEXVAL
0689 1D0A                   swapf   wreg
068A B50F                   andlw   0x0F
068B 4A4F                   movpf   wreg,HEXTMP
068C 2D0A                   negw    wreg
068D B109                   addlw   0x09
068E 970A                   btfss   wreg,MSB
068F C693                   goto    puth20
0690 B037                   movlw   'A'-0x0A
0691 0E4F                   addwf   HEXTMP,W
0692 C695                   goto    puth25
                    puth20
0693 B030                   movlw   '0'
0694 0E4F                   addwf   HEXTMP,W
                    puth25
0695 E679                   call    PutChar

0696 6A4E                   movfp   HEXVAL,wreg
0697 B50F                   andlw   0x0F
0698 4A4F                   movpf   wreg,HEXTMP
0699 2D0A                   negw    wreg
069A B109                   addlw   0x09
069B 970A                   btfss   wreg,MSB
069C C6A0                   goto    putl20
069D B037                   movlw   'A'-0x0A
069E 0E4F                   addwf   HEXTMP,W
069F C6A2                   goto    putl25
                    putl20
06A0 B030                   movlw   '0'
06A1 0E4F                   addwf   HEXTMP,W
                    putl25
06A2 E679                   call    PutChar

06A3 0002                   return

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME:         PutStr
                    ;
                    ; DESCRIPTION:  Sends a character string out the serial port.
                    ;

                    PutStr
06A4 AB4C                   tablrd  1,1,STRVALH
                    GetNextPair

06A5 A24C                   tlrd    1,STRVALH
06A6 A94D                   tablrd  0,1,STRVALL

06A7 6A4C                   movfp   STRVALH,wreg
06A8 31C1                   cpfseq  ZERO
06A9 C6AB                   goto    putH
06AA 0002                   return
                    putH
06AB E679                   call    PutChar

06AC 6A4D                   movfp   STRVALL,wreg
06AD 31C1                   cpfseq  ZERO
06AE C6B0                   goto    putL
06AF 0002                   return
                    putL
06B0 E679                   call    PutChar
06B1 C6A5                   goto    GetNextPair

                    ;*****************************************************************************

                    ;*****************************************************************************
```


© 1993 Microchip Technology Inc.

```
                        ; NAME:         GetHex
                        ;
                        ; DESCRIPTION:  Receive an ASCII hex character from the serial port and
                        ;               convert to numerical value.
                        ;
                        ; RETURNS:      numerical value in HEXVAL

                        GetHex

                        getnxt
06B2 E557                       call    IdleFunction
06B3 E681                       call    GetChk
06B4 31C2                       cpfseq  ONE
06B5 C6B2                       goto    getnxt

06B6 2950                       clrf    HEXSTAT
06B7 E676                       call    GetChar
06B8 4A4E                       movpf   wreg,HEXVAL
06B9 E679                       call    PutChar
06BA B00D                       movlw   CR
06BB 044E                       subwf   HEXVAL,W
06BC 330A                       tstfsz  wreg
06BD C6BF                       goto    gth10
06BE C6C9                       goto    gthCR
                        gth10

06BF 6A4E                       movfp   HEXVAL,wreg
06C0 B239                       sublw   '9'
06C1 970A                       btfss   wreg,MSB
06C2 C6C5                       goto    gth20

06C3 B009                       movlw   0x09
06C4 0F4E                       addwf   HEXVAL

                        gth20
06C5 B00F                       movlw   0x0F
06C6 0B4E                       andwf   HEXVAL
06C7 2950                       clrf    HEXSTAT
06C8 0002                       return

                        gthCR
06C9 B001                       movlw   0x01
06CA 4A50                       movpf   wreg,HEXSTAT
06CB 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         getval
                        ;
                        ; DESCRIPTION:  Get a value [800000,7FFFFF] from the serial port and place
                        ;               it in VALBUF.
                        ;
                                if      DECIO
                                else

                        GetVal
                                CLR24   VALBUF
                        getnext
                                call    GetHex

                                movlw   0x01
                                cpfseq  HEXSTAT
                                goto    shift
                                return
                        shift
                                swapf   VALBUF+B2
                                movfp   VALBUF+B2,wreg
```

```
                        andlw   0xF0
                        movpf   wreg,VALBUF+B2
                        swapf   VALBUF+B1
                        movfp   VALBUF+B1,wreg
                        andlw   0x0F
                        addwf   VALBUF+B2
                        movfp   VALBUF+B1,wreg
                        andlw   0xF0
                        movpf   wreg,VALBUF+B1
                        swapf   VALBUF+B0
                        movfp   VALBUF+B0,wreg
                        andlw   0x0F
                        addwf   VALBUF+B1
                        movfp   VALBUF+B0,wreg
                        andlw   0xF0
                        addwf   HEXVAL,W
                        movpf   wreg,VALBUF+B0

                        goto    getnext

                        endif

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME:          GetDecVal
                ;
                ; DESCRIPTION:   Get a value [-8388608,8388607] from the serial port and
                ;                place it in VALBUF
                ;
                ; RETURNS:       numerical value is returned in VALBUF

                        if     DECIO

                GetDecVal
                         CLR24   VALBUF

06CC 2931               CLRF    VALBUF+B0
06CD 2932               CLRF    VALBUF+B1
06CE 2933               CLRF    VALBUF+B2

06CF E708               call    GetDec
06D0 2B9B               setf    DECSIGN
06D1 B001               movlw   DEC_MN
06D2 3199               cpfseq  DECSTAT
06D3 299B               clrf    DECSIGN

                getdecnext
06D4 E708               call    GetDec

06D5 B002               movlw   DEC_CR
06D6 3199               cpfseq  DECSTAT
06D7 C6D9               goto    mul10
06D8 C6FD               goto    fixsign
                mul10

                        RLC24   VALBUF                  ; multiply VALBUF by two

06D9 8804               BCF     _carry
06DA 1B31               RLCF    VALBUF+B0
06DB 1B32               RLCF    VALBUF+B1
06DC 1B33               RLCF    VALBUF+B2

                        MOV24   VALBUF,DVALBUF          ; save in DVALBUF

06DD 6A31               MOVFP   VALBUF+B0,wreg          ; get byte of VALBUF into w
06DE 4A34               MOVPF   wreg,DVALBUF+B0         ; move to DVALBUF(B0)
06DF 6A32               MOVFP   VALBUF+B1,wreg          ; get byte of VALBUF into w
06E0 4A35               MOVPF   wreg,DVALBUF+B1         ; move to DVALBUF(B1)
```


© 1993 Microchip Technology Inc.

```
06E1 6A33                   MOVFP   VALBUF+B2,wreg        ; get byte of VALBUF into w
06E2 4A36                   MOVPF   wreg,DVALBUF+B2       ; move to DVALBUF(B2)

                            RLC24   VALBUF

06E3 8804                   BCF     _carry
06E4 1B31                   RLCF    VALBUF+B0
06E5 1B32                   RLCF    VALBUF+B1
06E6 1B33                   RLCF    VALBUF+B2

                            RLC24   VALBUF                ; VALBUF now multiplied by eight

06E7 8804                   BCF     _carry
06E8 1B31                   RLCF    VALBUF+B0
06E9 1B32                   RLCF    VALBUF+B1
06EA 1B33                   RLCF    VALBUF+B2

                            ADD24   DVALBUF,VALBUF        ; VALBUF now multiplied by ten

06EB 6A34                   MOVFP   DVALBUF+B0,wreg       ; get lowest byte of DVALBUF into w
06EC 0F31                   ADDWF   VALBUF+B0             ; add lowest byte of VALBUF, save in
06ED 6A35                   MOVFP   DVALBUF+B1,wreg       ; get 2nd byte of DVALBUF into w
06EE 1132                   ADDWFC  VALBUF+B1             ; add 2nd byte of VALBUF, save in
06EF 6A36                   MOVFP   DVALBUF+B2,wreg       ; get 3rd byte of DVALBUF into w
06F0 1133                   ADDWFC  VALBUF+B2             ; add 3rd byte of VALBUF, save in

                            CLR24   DVALBUF

06F1 2934                   CLRF    DVALBUF+B0
06F2 2935                   CLRF    DVALBUF+B1
06F3 2936                   CLRF    DVALBUF+B2
06F4 6A98                   movfp   DECVAL,wreg
06F5 4A34                   movpf   wreg,DVALBUF+B0
                            ADD24   DVALBUF,VALBUF
06F6 6A34                   MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
06F7 0F31                   ADDWF   VALBUF+B0       ; add lowest byte of VALBUF, save in
06F8 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
06F9 1132                   ADDWFC  VALBUF+B1       ; add 2nd byte of VALBUF, save in VALBUF(B1)
06FA 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
06FB 1133                   ADDWFC  VALBUF+B2       ; add 3rd byte of VALBUF, save in VALBUF(B2)

06FC C6D4                   goto    getdecnext
                fixsign
06FD 290A                   clrf    wreg
06FE 329B                   cpfsgt  DECSIGN
06FF 0002                   return
                            NEG24   VALBUF

0700 1331                   COMF    VALBUF+B0
0701 1332                   COMF    VALBUF+B1
0702 1333                   COMF    VALBUF+B2
0703 290A                   CLRF    wreg
0704 1531                   INCF    VALBUF+B0
0705 1132                   ADDWFC  VALBUF+B1
0706 1133                   ADDWFC  VALBUF+B2

0707 0002                   return

                            endif
```

4

© 1993 Microchip Technology Inc.

```
                        ;*****************************************************************************
                        ;*****************************************************************************
                        ; NAME:         GetDec
                        ;
                        ; DESCRIPTION:  Receive an ASCII decimal character from the serial port and
                        ;               convert to its numerical value.
                        ;
                        ; ARGUMENTS:    numerical value is returned in DECVAL
                        ;
                                if     DECIO

                        GetDec

                        getdecnxt
0708 E557                       call    IdleFunction
0709 E681                       call    GetChk
070A 31C2                       cpfseq  ONE
070B C708                       goto    getdecnxt

070C E676                       call    GetChar
070D 4A98                       movpf   wreg,DECVAL
070E E679                       call    PutChar

070F B00D                       movlw   CR
0710 0498                       subwf   DECVAL,W
0711 30C1                       cpfslt  ZERO
0712 C71F                       goto    gtdCR
0713 B02D                       movlw   MN
0714 0498                       subwf   DECVAL,W
0715 30C1                       cpfslt  ZERO
0716 C722                       goto    gtdMN
0717 B020                       movlw   SP
0718 0498                       subwf   DECVAL,W
0719 30C1                       cpfslt  ZERO
071A C725                       goto    gtdSP
                        gtd09
071B B00F                       movlw   0x0F
071C 0B98                       andwf   DECVAL
071D 2999                       clrf    DECSTAT
071E 0002                       return
                        gtdCR
071F B002                       movlw   DEC_CR
0720 4A99                       movpf   wreg,DECSTAT
0721 0002                       return
                        gtdMN
0722 B001                       movlw   DEC_MN
0723 4A99                       movpf   wreg,DECSTAT
0724 0002                       return
                        gtdSP
0725 B000                       movlw   DEC_SP
0726 4A99                       movpf   wreg,DECSTAT
0727 0002                       return

                                endif

                        ;*****************************************************************************


                        ;*****************************************************************************
                        ; NAME:         PutVal
                        ;
                        ; DESCRIPTION:  Sends the value in VALBUF [800000,7FFFFF] out the serial
                        ;

                                if     DECIO
                                else

                        PutVal
```


© 1993 Microchip Technology Inc.

```
                        movfp   VALBUF+B2,wreg
                        call    PutHex
                        movfp   VALBUF+B1,wreg
                        call    PutHex
                        movfp   VALBUF+B0,wreg
                        call    PutHex

                        return

                        endif
                        ;*****************************************************************************
                        ;*****************************************************************************
                        ; NAME:         PutDecVal
                        ;
                        ; DESCRIPTION:  Send the value in VALBUF [-8388608,8388607] out the serial
                        ;

                           if   DECIO

                    PutDecVal
0728 9733                   btfss   VALBUF+B2,MSB
0729 C734                   goto    pdpos
                            NEG24   VALBUF

072A 1331                   COMF    VALBUF+B0
072B 1332                   COMF    VALBUF+B1
072C 1333                   COMF    VALBUF+B2
072D 290A                   CLRF    wreg
072E 1531                   INCF    VALBUF+B0
072F 1132                   ADDWFC  VALBUF+B1
0730 1133                   ADDWFC  VALBUF+B2


0731 B02D                   movlw   MN
0732 E679                   call    PutChar
0733 C736                   goto    pddigits
                    pdpos
0734 B020                   movlw   SP
0735 E679                   call    PutChar

                    pddigits
0736 B08D                   movlw   (DEC_TABLE & 0xff)      ; DEC_TABLE LSB
0737 4A0D                   movpf   wreg,tblptrl
0738 B007                   movlw   page DEC_TABLE          ; DEC_TABLE MSB
0739 4A0E                   movpf   wreg,tblptrh

073A A934                   tablrd  0,1,DVALBUF+B0
                    readNextDec

073B A034                   tlrd    0,DVALBUF+B0            ; read entry from table
073C AB35                   tablrd  1,1,DVALBUF+B1
073D A936                   tablrd  0,1,DVALBUF+B2

073E 2B0A                   setf    wreg                    ; unitsposition if end of table
073F 3134                   cpfseq  DVALBUF+B0
0740 C742                   goto    getdigit
0741 C756                   goto    unitsposition
                    getdigit
0742 1534                   incf    DVALBUF+B0              ; restore to power of 10
0743 2B98                   setf    DECVAL                  ; set DECVAL to -1
                    inc
0744 1598                   incf    DECVAL                  ; increment DECVAL
                            SUB24   DVALBUF,VALBUF          ; check if in range

0745 6A34                   MOVFP   DVALBUF+B0,wreg         ; get lowest byte of DVALBUF into w
0746 0531                   SUBWF   VALBUF+B0               ; sub lowest byte of VALBUF, save in
```

```
0747 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0748 0332                   SUBWFB  VALBUF+B1      ; sub 2nd byte of VALBUF, save in VALBUF(B1)
0749 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
074A 0333                   SUBWFB  VALBUF+B2      ; sub 3rd byte of VALBUF, save in VALBUF(B2)

074B 9733                   btfss   VALBUF+B2,MSB
074C C744                   goto    inc

                            ADD24   DVALBUF,VALBUF  ; if so, correct VALBUF for next digit

074D 6A34                   MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
074E 0F31                   ADDWF   VALBUF+B0       ; add lowest byte of VALBUF, save in
074F 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0750 1132                   ADDWFC  VALBUF+B1       ; add 2nd byte of VALBUF, save in VALBUF(B1)
0751 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
0752 1133                   ADDWFC  VALBUF+B2       ; add 3rd byte of VALBUF, save in VALBUF(B2)

0753 6A98                   movfp   DECVAL,wreg     ; send DECVAL
0754 E685                   call    PutDec

0755 C73B                   goto    readNextDec     ; get next table entry

                    unitsposition
0756 6A31                   movfp   VALBUF+B0,wreg  ; units position value now in VALBUF
0757 E685                   call    PutDec

0758 0002                   return


                            endif

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ;
                    ;
                    ;       TABLES:


                              CMD_START CMD_TABLE

                              CMD_TABLE

                              CMD_DEF do_null,DO_NULL

0759 000D                   DATA    DO_NULL
075A 0570                   DATA    do_null

                              CMD_DEF do_move,DO_MOVE

075B 004D                   DATA    DO_MOVE
075C 0572                   DATA    do_move

                              CMD_DEF do_mode,DO_MODE

075D 004F                   DATA    DO_MODE
075E 0580                   DATA    do_mode

                              CMD_DEF do_setparameter,DO_SETPARAMETER

075F 0053                   DATA    DO_SETPARAMETER
0760 059C                   DATA    do_setparameter

                              CMD_DEF do_readparameter,DO_READPARAMETER

0761 0052                   DATA    DO_READPARAMETER
0762 05C3                   DATA    do_readparameter
```


© 1993 Microchip Technology Inc.

```
                        CMD_DEF do_shutter,DO_SHUTTER

0763 0043               DATA    DO_SHUTTER
0764 05ED               DATA    do_shutter

                        CMD_DEF do_readcomposition,DO_READCOMPOSITION

0765 0050               DATA    DO_READCOMPOSITION
0766 060D               DATA    do_readcomposition

                        CMD_DEF do_readcomvelocity,DO_READCOMVELOCITY

0767 0056               DATA    DO_READCOMVELOCITY
0768 0616               DATA    do_readcomvelocity

                        CMD_DEF do_readactposition,DO_READACTPOSITION

0769 0070               DATA    DO_READACTPOSITION
076A 061F               DATA    do_readactposition

                        CMD_DEF do_readactvelocity,DO_READACTVELOCITY

076B 0076               DATA    DO_READACTVELOCITY
076C 0628               DATA    do_readactvelocity

                        CMD_DEF do_externalstatus,DO_EXTERNALSTATUS

076D 0058               DATA    DO_EXTERNALSTATUS
076E 0631               DATA    do_externalstatus

                        CMD_DEF do_movestatus,DO_MOVESTATUS

076F 0059               DATA    DO_MOVESTATUS
0770 0638               DATA    do_movestatus

                        CMD_DEF do_readindposition,DO_READINDPOSITION

0771 0049               DATA    DO_READINDPOSITION
0772 063C               DATA    do_readindposition

                        CMD_DEF do_setposition,DO_SETPOSITION

0773 0048               DATA    DO_SETPOSITION
0774 0645               DATA    do_setposition

                        CMD_DEF do_reset,DO_RESET

0775 005A               DATA    DO_RESET
0776 0658               DATA    do_reset

                        CMD_DEF do_stop,DO_STOP

0777 0073               DATA    DO_STOP
0778 065B               DATA    do_stop

                if      _PICMASTER_DEBUG
                        CMD_DEF do_capture,DO_CAPTURE

0779 0063               DATA    DO_CAPTURE
077A 065E               DATA    do_capture

                endif
                        CMD_END

                ;                                       ;
077B 0000               DATA    0x00

                ;
                ; In PAR_TABLE, the code word is as follows :
```

```
                        ;       Low Byte is # of bytes, Hi Byte is function code
                        ;

077C 0003               PAR_TABLE       DATA    0x0003
077D 0020                               DATA    VL
077E 0103                               DATA    0x0103
077F 0023                               DATA    AL
0780 0202                               DATA    0x0202
0781 0026                               DATA    KP
0782 0302                               DATA    0x0302
0783 0028                               DATA    KV
0784 0402                               DATA    0x0402
0785 002A                               DATA    KI
0786 0501                               DATA    0x0501
0787 002C                               DATA    IM
0788 0602                               DATA    0x0602
0789 002D                               DATA    FV
078A 0702                               DATA    0x0702
078B 002F                               DATA    FA
078C 0008                               DATA    NUMPAR

                              if    DECIO

078D 423F               DEC_TABLE       DATA    0x423F
078E 000F                               DATA    0x000F
078F 869F                               DATA    0x869F
0790 0001                               DATA    0x0001
0791 270F                               DATA    0x270F
0792 0000                               DATA    0x0000
0793 03E7                               DATA    0x03E7
0794 0000                               DATA    0x0000
0795 0063                               DATA    0x0063
0796 0000                               DATA    0x0000
0797 0009                               DATA    0x0009
0798 0000                               DATA    0x0000
0799 FFFF                               DATA    0xFFFF
                              endif


                        endif

                        if      _PICMASTER_DEBUG
                                include "picmastr.asm" ; PIC-MASTER Debug (data capture) File

0060                    #define CaptureAddr     0x8000 ; addr for dummy Table Writes (to TRACE

                        ;*************************************************************************
                        ; NAME:         doCaptureRegs
                        ;
                        ; DESCRIPTION:  Captures Desired Register Values To PIC-MASTER Trace Buffer
                        ;               Intended for PICMASTER Demo/debug/servo tuning Purposes Only
                        ;               Capture The following registers to Trace Buffer by putting
                        ;               A Trace point on a TABLW instruction. Trace only 2nd Cycle
                        ;
                        ;               (a) POSERROR (position error : 16 bits)
                        ;               (b) VELERROR (velocity error : 16 bits)
                        ;               (c) MPOSITION (measured position value : 24 bits)
                        ;               (d) MVELOCITY (measured velocity value : 24 bits)
                        ;               (e) POSITION (commanded position : 24 bits)
                        ;               (f) VELOCITY (commanded velocity : 24 bits)
                        ;               (g) Y (output of servo loop : 32 bits)
                        ;               (h) YPWM (output value written to PWM : 10 bits)
                        ;
                        ;
                        ;
                        doCaptureRegs
                                                                ; !end! hdr !skip start!
079A B000                       movlw   (CaptureAddr & 0xff)
079B 010D                       movwf   tblptrl
079C B080                       movlw   CaptureAddr/256
```


© 1993 Microchip Technology Inc.

```
079D 010E                       movwf   tblptrh                 ; setup table pointer address

079E AC79                       tablwt  0,0,POSERROR+B0         ; dummy tablwt
079F A67A                       tlwt    1,POSERROR+B1           ; now table latch = 16 bits contents
of POSERROR
                        capPerr
07A0 AC79                       tablwt  0,0,POSERROR+B0         ; perform actual table write of
POSERROR

07A1 AC7C                       tablwt  0,0,VELERROR+B0
07A2 A67D                       tlwt    1,VELERROR+B1           ; capture Velocity error
                        capVerr
07A3 AC7C                       tablwt  0,0,VELERROR+B0

07A4 AC72                       tablwt  0,0,MPOSITION+B0
07A5 A673                       tlwt    1,MPOSITION+B1          ; capture measured position
                        capMpos
07A6 AC72                       tablwt  0,0,MPOSITION+B0

07A7 AC55                       tablwt  0,0,POSITION+B0
07A8 A656                       tlwt    1,POSITION+B1           ; capture commanded position
                        capPos
07A9 AC55                       tablwt  0,0,POSITION+B0

07AA AC75                       tablwt  0,0,MVELOCITY+B0
07AB A676                       tlwt    1,MVELOCITY+B1          ; capture measured velocity
                        capMvel
07AC AC75                       tablwt  0,0,MVELOCITY+B0

07AD AC58                       tablwt  0,0,VELOCITY+B0
07AE A659                       tlwt    1,VELOCITY+B1           ; capture commanded velocity
                        capVel
07AF AC58                       tablwt  0,0,VELOCITY+B0

07B0 AC88                       tablwt  0,0,YPWM+B0
07B1 A689                       tlwt    1,YPWM+B1               ; capture commanded velocity
                        capPwm
07B2 AC88                       tablwt  0,0,YPWM+B0

                                DEC16   CAPTMP

07B3 290A                       CLRF    wreg
07B4 07BE                       DECF    CAPTMP+B0
07B5 03BF                       SUBWFB  CAPTMP+B1

                                TFSZ16  CAPTMP

07B6 6ABE                       MOVFP   CAPTMP+B0,wreg
07B7 08BF                       IORWF   CAPTMP+B1,W
07B8 330A                       TSTFSZ  wreg

07B9 0002                       return

07BA 0001                       DATA    0x0001                  ; HALT instruction (avail only in

                        HaltTrace
07BB 29BB                       clrf    CAPFLAG
                                MOV16   CAPCOUNT,CAPTMP

07BC 6ABC                       MOVFP   CAPCOUNT+B0,wreg        ; get byte of CAPCOUNT into w
07BD 01BE                       MOVWF   CAPTMP+B0               ; move to CAPTMP(B0)
07BE 6ABD                       MOVFP   CAPCOUNT+B1,wreg        ; get byte of CAPCOUNT into w
07BF 01BF                       MOVWF   CAPTMP+B1                  ; move to CAPTMP(B1)
07C0 0002                       return
                        ;*****************************************************************************

                         end
                                END
Errors   :     0
Warnings :     0
```

© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

Implementation of the Data Encryption Standard Using PIC17C42

INTRODUCTION

In January 1977, The United States government adopted a product cipher developed by IBM® as its official encryption standard [1]. This algorithm, called the Data Encryption Standard (DES), has been adopted as a worldwide standard for data encryption by ISO (International Standards Organization) [2, 3]. This application note describes the implementation of the DES algorithm on PIC17C42.

THE DATA ENCRYPTION STANDARD

The DES algorithm is a substitution cipher which takes a block of 64 bits of input (plaintext) into a unique block of 64 bits of output (ciphertext), under the control of a 64 bit key, which is known only to the people intended to read the message. In this system, plaintext information is divided into several blocks which are then operated upon independently to generate a sequence of ciphertext blocks. The basic idea behind DES is to build a strong system out of simple, individually weak, components. The DES cryptosystem is based on a system of transpositions and permutations. The permutation box or P-box, is used to transpose, or map a sequence of input values to another sequence of values of the same length. Substitutions are performed by what is called S-boxes. A combination of the S-boxes and P-boxes can be viewed as a decoder/coder operation, where the output is simply a linear mapping of the input values. Each combination of the S-box and P-box comprises a single weak component of the algorithm. By including a sufficiently large number of stages in the product cipher, the output can be made to be a non-linear function of the input.

The mapping of input to output is one-to-one and invertible, since the encrypted messages can be decrypted. The DES has three distinct components: key schedule, cipher function and invertibility.

KEY SCHEDULE

The DES uses a 64-bit key for encryption and decryption process. Initially, the original 64-bit key is reduced to 56-bit by ignoring every eighth bit. In general these bits are used as parity bits to make sure that there were no errors when entering the key or during key transmission. After the 56-bit key is extracted a different 48-bit key, referred to as subkey, is generated for each of the 16 rounds of the DES. These keys, K_i, are determined as shown in Figure-1. The 56-bit key is divided into two 28-bit halves C_i and D_i which are then shifted left by either 1 or 2 digits, depending on the round. Table 1 shows the number of circular left shifts for C_i and D_i halves. After the shifting operation, 48 out of the 56 bits are selected. Since this operation permutes the order of the bits as well as selecting a subset of the original bits, it is called compression permutation or permuted choice. The permuted choice 1 and permuted choice 2 matrices are shown in Figure 2 and Figure 3 respectively.

TABLE 1: LEFT SHIFTS FOR KEY GENERATION

Iteration	# of left shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

4

© 1994 Microchip Technology Inc.

FIGURE 1 - KEY GENERATION

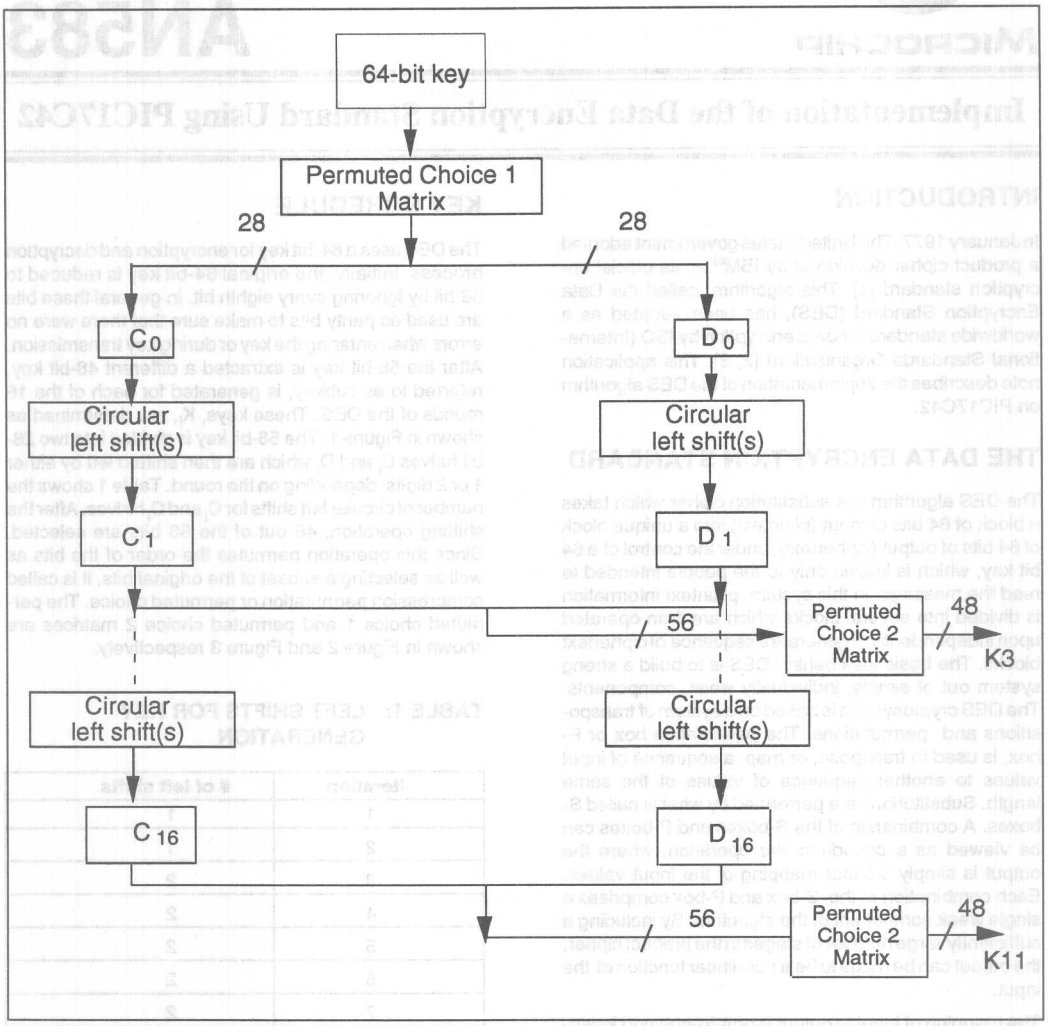


FIGURE 2 - PERMUTED CHOICE 1 MATRIX

57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37

FIGURE 3 - PERMUTED CHOICE 2 MATRIX

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

© 1994 Microchip Technology Inc.

CIPHER FUNCTION

The strength of the DES is based on the cipher function component. This is a fixed, highly non-linear function which guarantees that each bit of the ciphertext depend on every bit of the plaintext and every bit of the key.

After an initial permutation, the 64-bit block of plaintext is broken into a right half and left half, each 32 bits long. This step is followed by 16 identical rounds of operation, called function f, that combines the data with a 48-bit key, K_i. At each stage i, the inputs are the left block L_{i-1} and the right block R_{i-1} of the previous stage, and the outputs are the left shift block L_i and right block R_i of this stage. The outputs of L_i and R_i of each stage are computed from L_{i-1} and R_{i-1}, and a subkey K_i that is generated from the encryption key. In other words a round of the DES can be shown as:

$$L_i = R_{i-1}$$

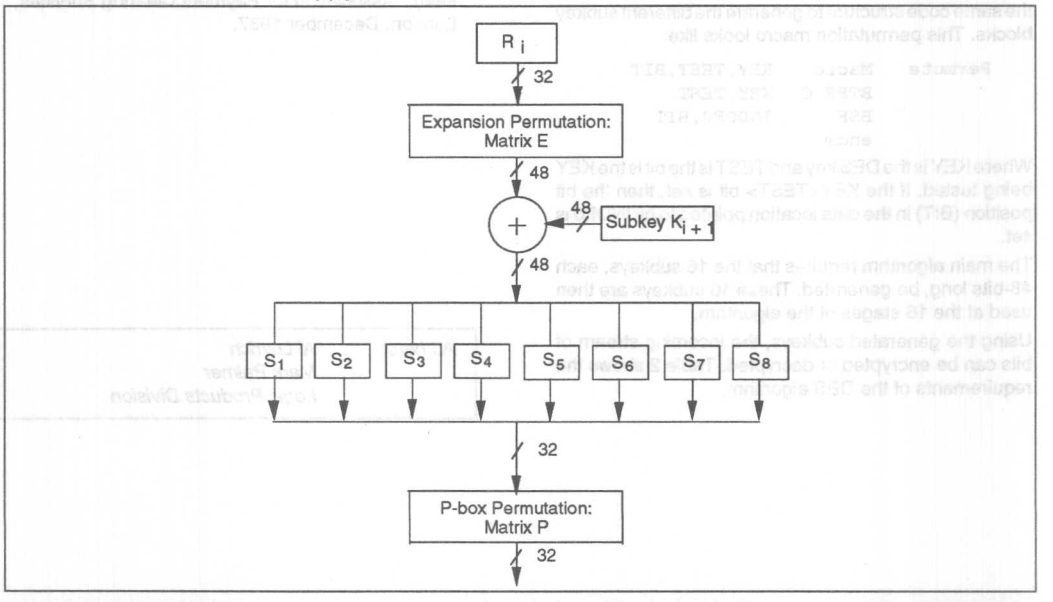
$$R_i = L_{i-1} \text{ XOR } f(R_{i-1}, K_i)$$

All the complexity of the DES algorithm lies in the function f, as shown in Figure-4. The function has four steps that are carried out in sequence. First a 48-bit number, E, is constructed by expanding the 32-bit previous right value, R_{i-1}, according to a fixed transposition and duplication rule. Then, K_i and R_i-1 are XORed together, generating a 48-bit result. This output is then partitioned into eight groups of 6 bits each, each of which is fed into a different S-box or substitution box. The S-boxes generate four instead of six outputs. In other words, the 64-bit input is mapped into a 32-bit output. Each S-box is a table of 4 rows and 16 columns. Each entry in the box is a 4-bit number. The six input bits of the S-box specify under which row and column number to look for the output. Figure-7 shows the 8 S-boxes.

The 6 input bits specify an entry in the S-box in a particular fashion as follows: the first and last bits of the sequence, taken together, represent a number between 0 and 3 (row entry), while the middle 4-bits represent a number between 0 and 15 (column entry). The output is simply the entry that corresponds to the (row, column) entry. For example, that the input to the first S-box is the binary sequence 110010. The first and last bits combine to form 10 which corresponds to the third row of the S-box. The middle four bits are combined to form 1001 which corresponds to the ninth column of the S-box. The corresponding entry in the first S-box is 12. Therefore the value of 1100 is substituted for 110010. The substitution boxes are the most critical step in the DES algorithm and more than anything else give DES its security.

Finally, the last stage consists of a permutation stage that generates a 32-bit output. After the 16 rounds, the left and right halves are joined, and a final permutation generates the ciphertext. The final permutation is the inverse of the initial permutation. Figure-8 shows a block diagram of the enciphering portion of the algorithm. While the reverse process of deciphering is shown in Figure 11. The initial permutation and inverse initial permutation matrices in Figure-8 are shown in Figures 9 and 10. Where the algorithm requires bit manipulation of a stream of data according to a matrix, the matrix is read from left to right, top to bottom, and interpreted as the bit position in the output block. For example, the initial permutation matrix transposes bit1 to bit58, bit2 to bit50, bit3 to bit42, etc.

FIGURE 4 - FUNCTION f (Ri, K_{i+1})



© 1994 Microchip Technology Inc.

INVERTIBILITY

The DES cipher function is not necessarily invertible, meaning to decode a message, it is not necessary to recover the input to the cipher function from its output and a knowledge of the key. In fact the cipher function must be highly non-linear to be resistant to plaintext attack (a method used for breaking a given algorithm). Invertibility of the DES is that one half-word of the output is precisely the bit configuration which was used to encode the other half, with the aid of the particular stage subkey. Therefore, by using the subkeys in reverse order, the encryption process can be reversed. This is really the reason that one half-word is always passed through unchanged - to provide the means of decrypting the other half-word.

PIC17C42 IMPLEMENTATION OF DES

CPU processing is required to generate the encryption key into the DES subkeys. The 64-bit encryption key is reduced to 56-bits, by ignoring every eighth bit, usually used as parity bit.

The majority of the DES code is for the Implementation of the permutation of the block of bits. The 56-bits of the key, stored in K1 through K8, scrambled-bit output is stored in the eight bytes D0-D7. The scrambling is accomplished by constructing D0-D7, one bit at a time. This is accomplished by initializing the D0 to D7 locations to a known state (cleared). Then the 64-bits of plain text are processed through the Initial Permutation Matrix (IP), which permutes the plain text and divides the information into two 32-bit blocks.

The use of Indirect addressing and the PIC16/17's single word instructions allows tight efficient coding of the DES algorithm. These bit testing capabilities allows the same code structure to generate the different subkey blocks. This permutation macro looks like:

```
Permute   Macro   KEY,TEST,BIT
          BTFS C  KEY,TEST
          BSF     INDOF0,BIT
          endm
```

Where KEY is the DES key and TEST is the bit is the KEY being tested. If the KEY<TEST> bit is set, then the bit position (BIT) in the data location pointed to by INDF0 is set.

The main algorithm requires that the 16 subkeys, each 48-bits long, be generated. These 16 subkeys are then used at the 16 stages of the algorithm.

Using the generated subkeys, the incoming stream of bits can be encrypted or decrypted. Table 2 shows the requirements of the DES algorithm.

TABLE 2: DES ALGORITHM REQUIREMENTS

Function	Program Memory (words)	Execution time Instruction cycles	ms
Key management and subkey generation	382	2729	0.436
Encryption	798	7714	1.234

A bit rate of about 51 kbps baud can be achieved, with a device utilization of 100%. This makes the PIC17C42 a price/performance leader for DES algoritms.

CONCLUSION

We have demonstrated the implementation of the DES algorithm on the PIC17C42 microcontroller. The 160 ns cycle time of the PIC17C42 makes possible a half-duplex rate of approximately 51 kbps for the DES. This rate is as good or superior to other implementations of the algorithm. The high performance of the PIC17C42 provides a low cost alternative to many dedicated solutions resulting in minimum system cost because of the programmability of the device.

References

1. NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, US Department of Commerce, January 1977.
2. ISO DIS 8730, "Banking Requirements for Message Authentication (Wholesale)," Association for Payment Clearing Services, London, July 1987.
3. ISO DIS 8732, "Banking Key Management (Wholesale)," Association for Payment Clearing Services, London, December 1987.

Authors: *Al Lovrich*
Mark Palmer
Logic Products Division

© 1994 Microchip Technology Inc.

FIGURE 5 - MATRIX E

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

FIGURE 6 - MATRIX P

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

FIGURE-7 - S MATRICES

```
/* S1 */
14  4 13  1  2 15 11  8  3 10  6 12  5  9  0  7
 0 15  7  4 14  2 13  1 10  6 12 11  9  5  3  8
 4  1 14  8 13  6  2 11 15 12  9  7  3 10  5  0
15 12  8  2  4  9  1  7  5 11  3 14 10  0  6 13

/* S2 */
15  1  8 14  6 11  3  4  9  7  2 13 12  0  5 10
 3 13  4  7 15  2  8 14 12  0  1 10  6  9 11  5
 0 14  7 11 10  4 13  1  5  8 12  6  9  3  2 15
13  8 10  1  3 15  4  2 11  6  7 12  0  5 14  9

/* S3 */
10  0  9 14  6  3 15  5  1 13 12  7 11  4  2  8
13  7  0  9  3  4  6 10  2  8  5 14 12 11 15  1
13  6  4  9  8 15  3  0 11  1  2 12  5 10 14  7
 1 10 13  0  6  9  8  7  4 15 14  3 11  5  2 12

/* S4 */
 7 13 14  3  0  6  9 10  1  2  8  5 11 12  4 15
13  8 11  5  6 15  0  3  4  7  2 12  1 10 14  9
10  6  9  0 12 11  7 13 15  1  3 14  5  2  8  4
 3 15  0  6 10  1 13  8  9  4  5 11 12  7  2 14

/* S5 */
 2 12  4  1  7 10 11  6  8  5  3 15 13  0 14  9
14 11  2 12  4  7 13  1  5  0 15 10  3  9  8  6
 4  2  1 11 10 13  7  8 15  9 12  5  6  3  0 14
11  8 12  7  1 14  2 13  6 15  0  9 10  4  5  3

/* S6 */
12  1 10 15  9  2  6  8  0 13  3  4 14  7  5 11
10 15  4  2  7 12  9  5  6  1 13 14  0 11  3  8
 9 14 15  5  2  8 12  3  7  0  4 10  1 13 11  6
 4  3  2 12  9  5 15 10 11 14  1  7  6  0  8 13

/* S7 */
 4 11  2 14 15  0  8 13  3 12  9  7  5  0  6  1
13  0 11  7  4  9  1 10 14  3  5 12  2 15  8  6
 1  4 11 13  2  3  7 14 10 15  6  8  0  5  9  2
 6 11 13  8  1  4 10  7  9  5  0 15 14  2  3 12

/* S8 */
13  2  8  4  6 15 11  1 10  9  3 14  5  0 12  7
 1 15 13  8 10  3  7  4 12  5  6 11  0 14  9  2
 7 11  4  1  9 12 14  2  0  6 10 13 15  3  5  8
 2  1 14  7  4 10  8 13 15 12  9  0  3  5  6 11
```

FIGURE 9 - INITIAL PERMUTATION MATRIX

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

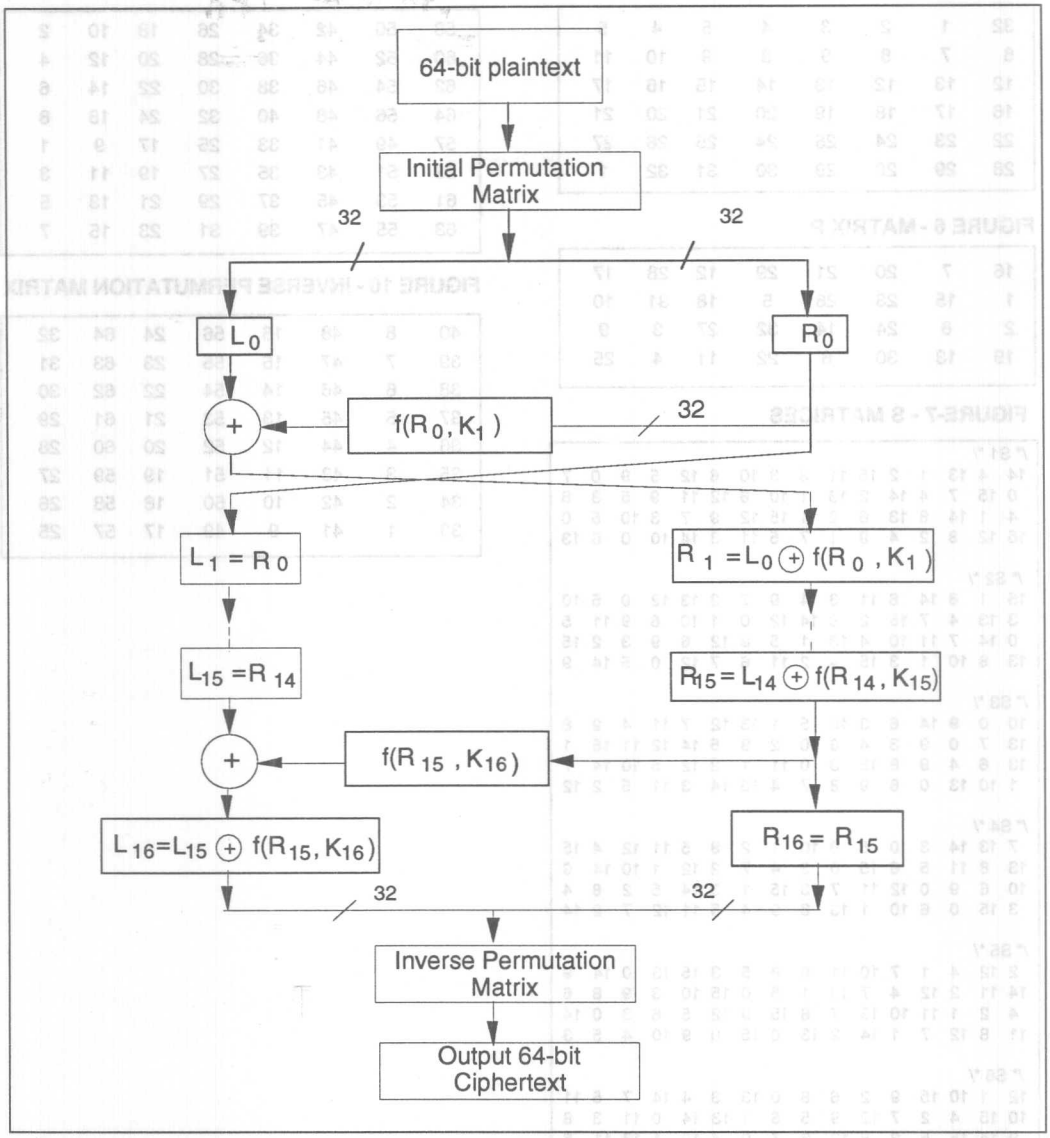
FIGURE 10 - INVERSE PERMUTATION MATRIX

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

4

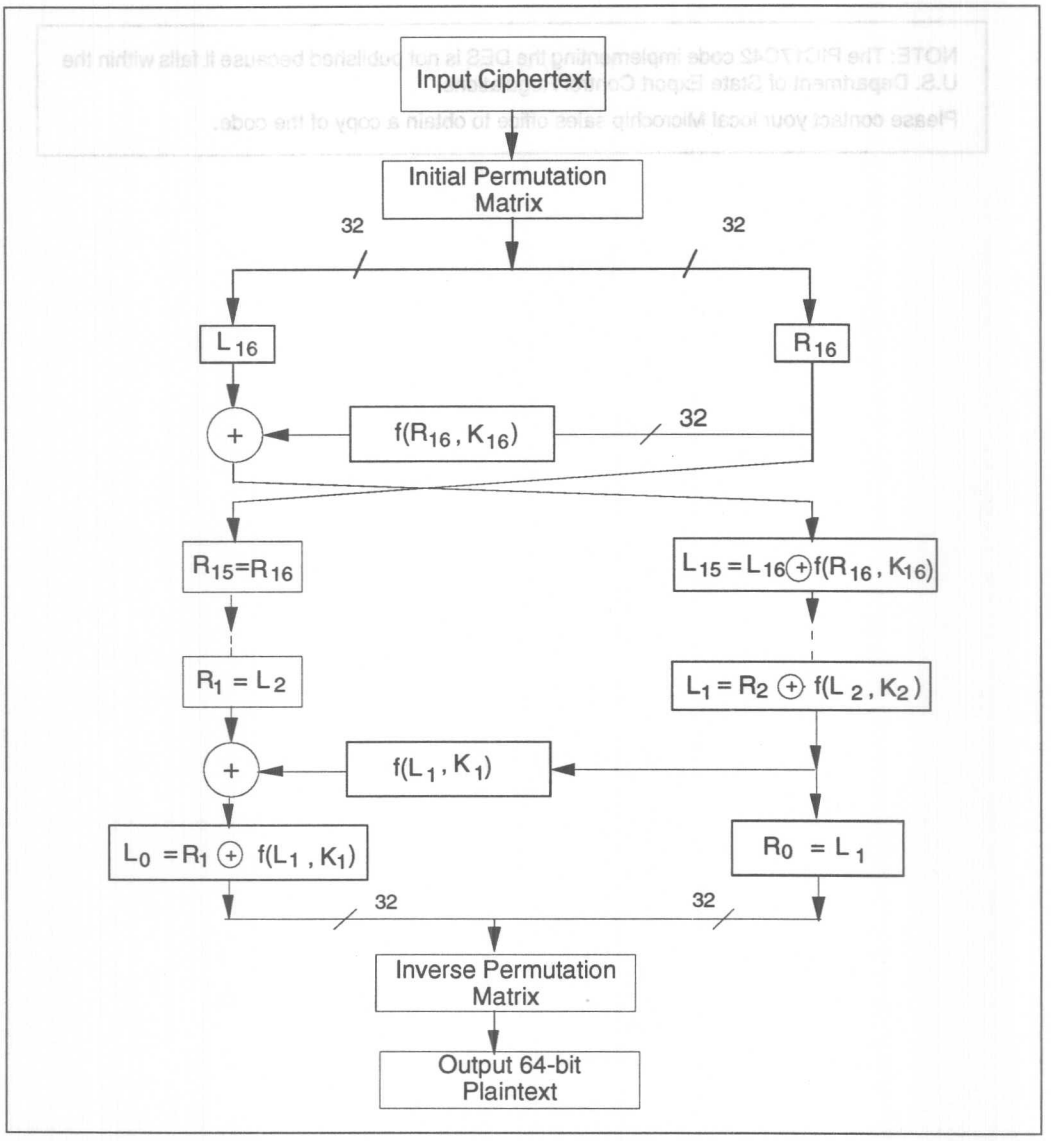
© 1994 Microchip Technology Inc.

FIGURE 8 - DES ENCRYPTION BLOCK DIAGRAM



© 1994 Microchip Technology Inc.

FIGURE 11 - DES DECRYPTION BLOCK DIAGRAM

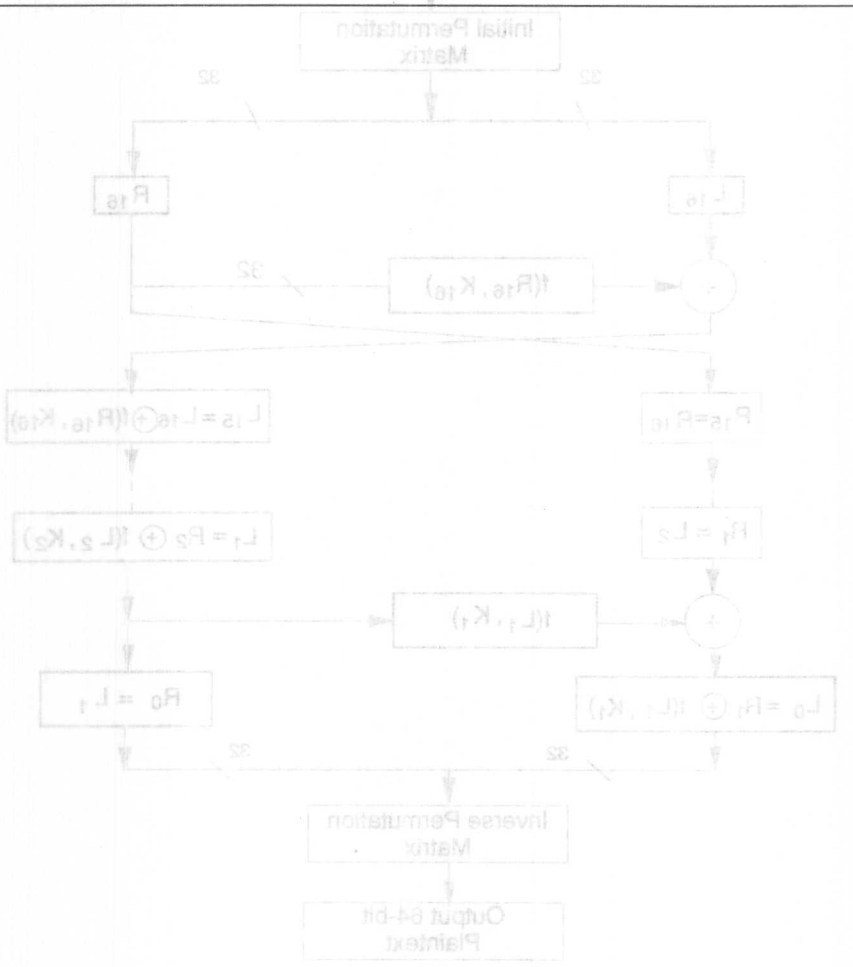


4

APPENDIX A

NOTE: The PIC17C42 code implementing the DES is not published because it falls within the U.S. Department of State Export Control Regulations.

Please contact your local Microchip sales office to obtain a copy of the code.



© 1994 Microchip Technology Inc.

SECTION 5
PIC16/17 APPLICATION NOTES

Implementing a Table Read - AN556 ..... 5- 1
Techniques to Disable Global Interrupts - AN576 ..... 5- 5
IEEE 754 Compliant Floating-Point Routines - AN575 ..... 5- 11
PIC16C5X / 16CXX Utility Math Routines - AN526 ..... 5- 89
A Real-Time Operating System for PIC16/17 - AN585 5-167
PIC16/17 Oscillator Design Guide - AN588 5-213
PICMASTER™ Support of Microsoft® Windows™ DDE - AN584 5-231

5

© 1994 Microchip Technology Inc.

Implementing a Table Read

INTRODUCTION

This application note shows how to implement a Table look-up for all PIC16/17 products. The examples shown are for the PIC16CXX family. An explanation of differences for the PIC16C5X family and PIC17C42 is at the end of this application note.

To access data in Program Memory, a table read operation must be performed. The Table consists of a series of `RETLW K` instructions where, the 8-bit table constants are assigned to the literal K. The first instruction in the Table computes the offset to the table by using "addwf pcl" and consequently the program branches to the appropriate `retlw X` instruction (see Example 1).

Example 1.

```
    .
    .
    movlw  offset ;load offset in w reg
    call   Table
    .
    .
    .
Table:
    addwf  pcl    ;add offset to pc to
                  ;generate a computed goto
    retlw  'A'    ;return the ASCII char A
    retlw  'B'    ;return the ASCII char B
    retlw  'C'    ;return the ASCII char C
    .
    .
    .
```

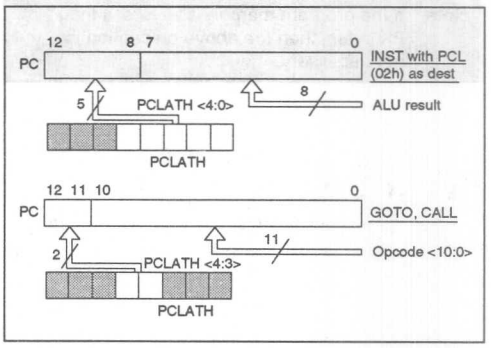
The method is straight forward, however certain precautions have to be exercised when doing a Table read in the PIC16CXX.

IMPLEMENTATION

Program Counter Loading

The PC in the PIC16CXX is 13-bits wide. The low 8-bits (PCL) are mapped in RAM at location 02 and are directly readable and writable. The high 5-bits are not accessible directly and can only by written through the PCLATH (see Figure 1). The PCLATH is a r/w register with only five of its bits implemented <4:0>, all other bits are read as '0'.

FIGURE 1 - LOADING OF PC IN DIFFERENT SITUATIONS



5

© 1994 Microchip Technology Inc.

SECTION 1

Call and Goto Instructions

When executing a call or goto, the low 11-bits are loaded directly from the instruction opcode. The high 2-bits are loaded from bits 3 and 4 of the PCLATH. It is a good practice to pre-load the PCLATH with the high byte of the location of the routine before executing the routine. This can be done as follows:

Example 2.

```
    .
    .
    .
    movlw   HIGH Table      ;load high 8 bit
                            ;address of Table
    movwf   PCLATH          ;into PCLATH
    call    Routine         ;execute Call
                            ;instruction
    .
    .
    .
```

Note: If the program memory size is less than 2K bytes, then the above precaution is not necessary.

Computed Goto Instruction

Any instruction with PCL as the destination, will load the PCH with the 5 low bits from the PCLATH (see Figure 1). In Program 3, if the address where the call was made, was on page 0 and the address of the actual table was on page 3, then when executing the computed goto, the program will go to a location in page 0 instead of a location on page 3. To avoid the program from branching "erratically" when doing a table read, the PCLATH should be pre-loaded with the high byte of the "Table" address. Example 3 shows how this can be done.

Example 3.

```
    .
    org     0x80    ;code location in page 0
    movlw   offset  ;load offset in w reg
    call    Table
    .
    .
    org     0x0320  ;Table located in page 3
Table
    addwf   pcl     ;add offset to pc to
                    ;generate a computed goto
    retlw   'A'     ;return the ASCII char A
    retlw   'B'     ;return the ASCII char B
    retlw   'C'     ;return the ASCII char C
    .
    .
    .
```


© 1994 Microchip Technology Inc.

When doing a computed goto for a table read, care should taken about page boundaries. The ADDWF PCL instruction will not compute a value greater than 8-bits. In Example 4, the result of the computed goto will result in a branch to an unintended portion of the code. The user either has to be cautious as to where in a page the Table is resident or will have to monitor page roll-over and add it to the PCLATH ahead of the computed goto.

Example 4.

```
  .
  org    0x80        ;code location in
                      ;page 0
  movlw  HIGH Table  ;load PCLATH with hi
                      ;address
  movwf  PCLATH      ;   /
  movlw  offset      ;load offset in w reg
  call   Table
  .
  .
  org    0x02ff      ;Table located end of
                      ;page 2
Table:
  addwf  pcl         ;value in pc will not
                      ;roll over to page 3
  retlw  'A'         ;return the ASCII
                      ;char A
  retlw  'B'         ;return the ASCII
                      ;char B
  retlw  'C'         ;return the ASCII
                     ;char C
  .
  .
  .
```

To take care of both table location and page boundary crossing, it is necessary to do a 13-bit computed goto operation as shown in Example 5.

The code in Example 5 will allow the user to place and access a table anywhere in program memory.

Example 5

```
    .
    movlw  LOW Table ;get low 8 bits of
                      ;address
    addwf  offset    ;do a 8 bit add
                      ;operation
    movlw  HIGH Table ;get high 5 bits of
                      ;address
    btfsc  status,c  ;page crossed?
    addlw  1         ;yes then inc high
                      ;address
    movwf  PCLATH    ;load high address in
                      ;latch
    movf   offset,w  ;load computed offset
                      ;in w reg
    call   Table
    .
    .
Table:
    movwf  pcl       ;load computed offset
                      ;in PCL
    retlw  'A'       ;return the ASCII
                      ;char A
    retlw  'B'       ;return the ASCII
                      ;char B
    retlw  'C'       ;return the ASCII
                      ;char C
    .
    .
    .
```

Interrupts

Interrupts are handled like a call, i.e. the present PC+1 address is saved on the stack and the program vectors to location 0x04. In example 5, if the interrupt occurs just before the "movwf pcl" instruction (at label "Table"), then the present fetched instruction will be executed, i.e. movwf pcl, the new computed PC will be incremented and saved on stack (as the return from interrupt address) and the program will vector to the interrupt vector. On return from interrupt, the program will go to the intended offset of the table + 1. This is a very undesirable result, so interrupts must be disable during a table read operation.

© 1994 Microchip Technology Inc.

SECTION 2

Implementation for the PIC16C5X Family

The PIC16C5X has no PCH or PCLATH register, so the user has to take into consideration all the precautions mentioned in Section 1. In PIC16C5X, the location of the Table has to be in the top-half of a 512 byte page. This restriction is not valid for the PIC16CXX family. To convert a table read operation from PIC16C5X code to the PIC16CXX code, the following should be done:

1. Remove any program memory bank select instructions (PIC16C56/57), if present. These are not necessary for the PIC16CXX.
2. Do a 13-bit computed goto operation (as shown in Example 5), when doing a table read operation.

Implementation for the PIC17C42

Unlike the PIC16CXX family, the PIC17C42 loads the PCLATH with the PCH value during a call or goto operation, however a computed goto, does not take care of page boundary crossing. A 16-bit computed jump address should be calculated before the table read is executed. Example 6 shows how this is done.

Example 6 allows the user to locate his Table at any program memory location, however, for large table or tables which cross page boundaries, it is recommended that the "tablrd / tlrd" instruction be used, these instructions are specific for table read operations and are very code efficient.

As mentioned in Section 1, interrupts must be disabled during a table read using the "retlw K" instruction (Example 6). Note it is not necessary to disable interrupts when using the "tablrd/tlrd" instruction for table read operations.

Example 6

```
  .
  movlw   LOW Table   ;get low 8 bits of
                        ;address
  addwf   offset      ;do a 8 bit add
                        ;operation
  movlw   HIGH Table  ;get high 8 bits of
                        ;address
  btfsc   status,c    ;page crossed?
  addlw   1           ;yes then inc high
                        ;address
  movwf   PchBuffer   ;load in  temp
                        ;location
  call    Table
  .
  .
Table:
  movf    PchBuffer,w ;get high offset
  movwf   PCLATH      ;load in latch
  movf    offset,w    ;get low offset
  movwf   pcl         ;load computed
                        ;offset in PCL
  retlw   'A'         ;return the ASCII
                        ;char A
  retlw   'B'         ;return the ASCII
                        ;char B
  retlw   'C'         ;return the ASCII
                        ;char C
  .
  .
  .
```

Author: Stan D'Souza, Logic Products Division

© 1994 Microchip Technology Inc.

Techniques to Disable Global Interrupts

This application brief discusses four methods for disabling global interrupts. The method best suited for the application may then be used. All discussion will be specific to the PIC16CXX family of products, but these concepts are also applicable to the PIC17C42, and are shown in the B examples.

To disable interrupts, either the Global Interrupt Enable (GIE) bit must be cleared or all the individual interrupt enable bits must be cleared. An issue arises when an instruction clears the GIE bit and an interrupt occurs "simultaneously". For example, when a program executes the instruction BCF INTCON, GIE (at address PC), there is a possibility that an interrupt will occur during this instruction. If an interrupt occurs during this instruction, the program would complete execution of this instruction, and then immediately branch to the user's interrupt service routine. This occurs because the GIE bit was not clear (disabled) when the interrupt occurred. Normally at the end of the interrupt service routine is the RETFIE instruction. This instruction causes the program to return to the instruction at PC + 1, but also sets the GIE bit (enabled). Therefore the GIE bit is not cleared as expected, and unintended program execution may occur.

One method to ensure that the GIE bit is cleared is shown in Example 1, as well as in the PIC16CXX data sheets. This method tests the state of the GIE bit, after clearing, to ensure that it was not accidentally set in the user's interrupt service routine by the RETFIE instruction. If the GIE bit was accidentally set, the program branches back to the instruction that clears the GIE bit.

In this method, the time to ensure that the GIE bit is cleared is indeterminate. Depending on the frequency of the enabled interrupts during this code segment, unexpected delays into the following code segment may occur. For some applications, this may be undesirable. The following three methods address this issue.

5

EXAMPLE 1A: FORCING THE GIE BIT CLEAR (METHOD 1, PIC16CXX)

```
        :
LOOP    BCF     INTCON, GIE         ; Disable Global Interrupts
        BTFSC   INTCON, GIE         ; Global Interrupts Disabled?
        GOTO    LOOP                ; NO, try again
        :                           ; YES, continue with program flow
        :
        BSF     INTCON, GIE         ; Re-enable Global Interrupts
```

EXAMPLE 1B: FORCING THE GLINTD BIT CLEAR (METHOD 1, PIC17C42)

```
        :
LOOP    BSF     CPUSTA, GLINTD      ; Disable Global Interrupts
        BTFSS   CPUSTA, GLINTD      ; Global Interrupts Disabled?
        GOTO    LOOP                ; NO, try again
        :                           ; YES, continue with program flow
        :
        BSF     INTCON, GIE         ; Re-enable Global Interrupts
```

© 1994 Microchip Technology Inc.

The second method is to disable the individual interrupt enable bits. If it is known which bits are enabled at this point, it can easily be done. Example 2 shows the disabling of interrupts, where it is known which sources are enabled (some peripheral interrupts and the T0CKI pin interrupt).

This method also requires the same number of instructions for the disabling/enabling of interrupts as method 1, but requires a knowledge of which individual interrupt enable bits need to be disabled and (more importantly) re-enabled. The major advantage of this method is that it can minimizes the time delay entering the code segment which follows the point where interrupts are disabled.

EXAMPLE 2A: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC16CXX)

```
:
MOVLW  b'10011111'      ; Disable  Peripheral and T0CKI pin interrupts,
ANDWF  INTCON, F        ; All other bits unchanged
:
:
:
MOVLW  b'01100000'      ; Re-enable Peripheral and T0CKI pin interrupts,
IORWF  INTCON, F        ; All other bits unchanged
:
```

EXAMPLE 2B: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC17C42)

```
:
MOVLW b'11110011'       ; Disable Peripheral and T0CKI pin interrupts,
ANDWF INTSTA, F         ; All other bits unchanged
:
:
:
MOVLW b'00001100'       ; Re-enable Peripheral and RT pin interrupts,
IORWF INTSTA, F         ; All other bits unchanged
:
```


© 1994 Microchip Technology Inc.

Method 3 can be used if the states of the individual interrupt enable bits are unknown. A temporary byte of data RAM is required to store the value of the INTCON register. This method is shown in Example 3.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, and also a byte of data RAM to temporarily store the value of the INTCON register. The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled.

EXAMPLE 3A: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC16CXX)

```
        :
MOVF    INTCON, W       ; Move the value in INTCON to
MOVWF   S_INTCON        ; a shadow register
MOVLW   b'10000111'     ; Disable all individual interrupts,
ANDWF   INTCON, F       ; All other bits unchanged
        :
        :
        :
MOVF    S_INTCON, W     ; Restore the INTCON register
IORWF   INTCON, F       ;
        :
```

EXAMPLE 3B: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC17C42)

```
        :
MOVPF   INTSTA, S_INTSTA ; Move the value in INTSTA to a shadow register
MOVLW   b'11110000'      ; Disable all individual interrupts,
ANDWF   INTSTA, F        ; All other bits unchanged
        :
        :
        :
MOVF    S_INTSTA, W      ; Restore the INTSTA register
IORWF   INTSTA, F        ;
        :
```

5

© 1994 Microchip Technology Inc.

The final method is to use a RAM location to "shadow" the value of the GIE bit. This shadow bit can then be used in the interrupt service routine to determine which return instruction to use. That is, either one of the RETURN or the RETFIE (which enables the GIE) instructions. Example 4 shows this implementation, which require that a general purpose bit be available to hold the "shadow" GIE value. In this example, the shadow GIE (S_GIE) bit is contained in the register FLAG_REG. If an interrupt occurs during the clearing of the shadow GIE, the interrupt is responded to. At the end of the interrupt service routine, the shadow GIE bit is cleared so the RETURN instruction is executed. The GIE bit remains disabled and program execution returns to the instruction which tries to clear the GIE (disable). No interrupts can occur during this instruction since the GIE bit was not re-enabled after the interrupt service routine.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, a single bit of data RAM to temporarily store the value of the desired GIE value, and increases the interrupt service routine execution time by one instruction cycle, for most occurrences of interrupts (two cycles worst case). The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled. Also, the individual interrupt enable bits need not be modified.

EXAMPLE 4A: THE "SHADOW" GIE BIT (METHOD 4, PIC16CXX)

```
        :
              org     0x004
INT_SERVICE_ROUTINE
        :
        :
        BTFSC  FLAG_REG, S_GIE      ; Is the S_GIE bit enabled?
        RETFIE                      ; YES, the GIE should be enabled
        RETURN                      ; NO, the GIE should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN    :
        :
        :
        BCF    FLAG_REG, S_GIE      ; Disable the shadow GIE bit
        BCF    INTCON, GIE          ; Disable the GIE bit
        :
        :
        :
        BSF    FLAG_REG, S_GIE      ; Enable the shadow GIE bit
        BSF    INTCON, GIE          ; Enable the GIE bit
        :
        :
        END
```

EXAMPLE 4B: THE "SHADOW" GLINTD BIT (METHOD 4, PIC17C42)

```
        :
              org     0x004
INT_SERVICE_ROUTINE
        :
        :
        BTFSS  FLAG_REG, S_GLINTD     ; Is the S_GLINTD bit enabled?
        RETFIE                        ; YES, the GLINTD should be enabled
        RETURN                        ; NO, the GLINTD should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN    :
        :
        :
        BSF    FLAG_REG, S_GLINTD     ; Disable the shadow GLINTD bit
        BSF    CPUSTA, GLINTD         ; Disable the GLINTD bit
        :
        :
        :
        BCF    FLAG_REG, S_GLINTD     ; Enable the shadow GLINTD bit
        BCF    CPUSTA, GLINTD         ; Enable the GLINTD bit
        :
        :
        END
```


© 1994 Microchip Technology Inc.

In conclusion, different methods exist to ensure that all interrupts are disabled. The requirement(s) of the applications determine which of the methods is the best fit. A comparison of the different methods is shown in Table 1.

TABLE 1: COMPARISON OF DIFFERENT METHODS

	Program Memory	Data Memory	Cycle Delay (Tcy) Best Case	Worst Case
Method 1	2 words * N	---	2	Indeterminate
Method 2	2 words * N	---	1	1 + Tisr
Method 3 - PIC16CXX	4 words * N	1 byte	3	3 + Tisr
- PIC17C42	3 words * N	1 byte	2	2 + Tisr
Method 4	2 words * N + 2 words	1 bit	1♣	1 + (Tisr + 2)

Legend: N - Number of occurances to disable / re-enable interrupts.

Tisr - Time to execute the interrupt service routine.

♣ This method increases the interrupt service routine time (Tisr) by 1 cycle for most occurances (2 cycles worst case).

5

Author: Mark Palmer - Logic Products Division
Contributions by Martin Burghardt - Manager Applications (Central Europe)

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN575

IEEE 754 Compliant Floating-Point Routines

Author: Frank Testa

INTRODUCTION

This application note presents an implementation of the following floating point math routines for the PIC16/17 microcontroller family:

INTxx(A)	float to integer conversion
FLOxx(A)	integer to float conversion
NRMxx(A)	normalize
FPAxx(A , B)	add/subtract
FPMxx(A , B)	multiply
FPDxx(A , B)	divide

Routines for the PIC16/17 families are provided in a modified IEEE754 32 bit format together with versions in 24-bit truncated format.

FLOATING POINT ARITHMETIC

Although fixed point arithmetic can usually be employed in many numerical problems through the use of proper scaling techniques, this approach can become complicated and sometimes result in less efficient code than is possible using floating point methods[1]. Floating point arithmetic is essentially equivalent to arithmetic in scientific notation relative to a particular base or radix. In the special case of base two or binary arithmetic, a number A has the floating point representation:

$$A = f * (2^e), \quad f = \sum_{k=1}^{n} \{ a(k) * (2^{-k}) \}$$

where f is the fraction or mantissa, e is the exponent or characteristic, n is the number of bits in f and a(k), k=1,..,n are the bit values of f with a(1)=MSB. Typically, the mantissa is in normalized sign-magnitude representation with implicit MSB equal to one, and e is stored in biased form, where the bias is the magnitude of the most negative possible exponent[1,2]. If m is the number of bits in the biased exponent e^b:

$$e^b = e + 2^{m-1}$$

Using biased exponents permits comparison of exponents through a simple unsigned comparator, and further results in a unique representation of zero given by f = eb = 0.

Algorithms for radix conversion are discussed in APPENDIX A, and can be used to produce the binary floating point representation of a given decimal number. Examples of sign-magnitude floating point representations of some decimal numbers are as follows:

Decimal	e	f
1.0	1	.10000000
0.15625	-2	.10100000
0.1	-3	.110011001100....
1.23×10^3	11	.10011001110

It is important to note that the only numbers that can be represented exactly in binary arithmetic are those which are sums of powers of two, resulting in non-terminating binary representations of some simple decimal numbers such as .1 as shown above, and leading to truncation errors regardless of the value of n. Floating point calculations, even involving numbers admitting an exact binary representation, usually lose information after truncation to an n bit result, and therefore require some rounding scheme to minimize such roundoff errors[1].

5

© 1994 Microchip Technology Inc.

ROUNDING METHODS

Truncation of a binary representation to n bits is severely biased since it always leads to a number whose magnitude is less than or equal to that of the exact value, thereby possibly causing significant error buildup during a long sequence of calculations. Simple adder-based rounding by adding the NSB to the LSB is unbiased except when the value to be rounded is equidistant from the two nearest n bit values[1]. In this case, magnitudes are always rounded up thereby producing a small but still undesirable bias. This can be removed by stipulating that in the equidistant case, the n bit value with LSB=0 is selected, commonly referred to as the rounding to the nearest method, the default mode in the IEEE754 standard[4,5]. The number of guard bits or extra bits of precision, is related to the sensitivity of the rounding method since using more guard bits results in fewer equidistant cases to be resolved. Since more than one guard bit requires an extra byte in PIC16/17 arithmetic, only one guard bit, usually handled in the carry bit, is employed in this library of floating point routines. Nearest neighbor rounding with one guard bit leads to the following simple result:

n Bit Value	Guard Bit	Result
A	0	round to A
A	1	if A,LSB=0, round to A if A,LSB=1, round to A+1
A+1	0	round to A+1

Another interesting rounding method, is Von Neumann rounding or jamming, where the exact number is truncated to n bits and then set LSB=1. Although the errors can be twice as large as in round to the nearest, it is unbiased and requires little more effort than truncation[1].

FLOATING POINT FORMATS

In what follows, we use the following floating point formats:

	eb	radix point	f0	f1	f2
IEEE754 32-bit	xxxxxxxx	.	Sxxxxxxx	xxxxxxxx	xxxxxxxx
truncated 24-bit	xxxxxxxx	.	Sxxxxxxx	xxxxxxxx	

where eb is the biased 8-bit exponent, with bias=2^7=128=0x80, S is the sign bit, and bytes f0, f1 and f2 constitute the mantissa with f0 the most significant byte with implicit MSB = 1. It is important to note that the IEEE754 standard format[4] places the sign bit as the MSB of eb with the LSB of the exponent as the MSB of f0. Because of the inherent byte structure of the PIC16/17 family of microcontrollers, more efficient code was possible by adopting the above formats rather than strictly adhering to the IEEE standard.

The limiting absolute values of the above floating point formats are given as follows:

	32-bit format	eb	IAI e	f	decimal
MAX	0xFF7FFFFF	FF	7F	7FFFFF	1.7014117E+38
MIN	0x01000000	01	81	000000	2.9387359E-39

where the MSB is implicitly equal to one, and its bit location is occupied by the sign bit. The 24-bit format has the same structure but with only a 16-bit mantissa. While 24- to 32-bit conversion is trivial, requiring only an additional zero byte in the mantissa, a 32- to 24-bit conversion routine would typically employ nearest neighbor rounding before truncation.

To produce the correct representation of a particular decimal number, a high-level language compiler and debugger could be used to display the internal binary representation on a host computer and make the appropriate conversion to the above format. If this approach is not feasible, algorithms for producing this representation are contained in Appendix A.

© 1994 Microchip Technology Inc.

FLOATING POINT EXCEPTIONS

Although the dynamic range of mathematical calculations is increased through floating point arithmetic, overflow and underflow are both possible when the limiting values of the representation are exceeded, such as in multiplication requiring the addition of exponents, or in division with the difference of exponents[2]. In these operations, mantissa calculations followed by appropriate normalizing and exponent modification can also lead to overflow or underflow in special cases. Similarly, addition and subtraction after mantissa alignment, followed by normalization can also lead to such exceptions. Furthermore, if nearest neighbor rounding is enabled, floating point overflow can occur in the case of rounding 0xFF7FFFFF with NSB=1 since adding the NSB would be required, resulting in mantissa overflow followed by a right shift and increment of the exponent, thus producing exponent overflow.

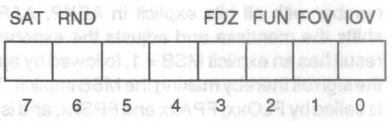
DATA RAM REQUIREMENTS

The following contiguous data ram locations are used by the library:

EXP	=	AEXP	AARG and ACC exponent
ACC+B0	=	AARG+B0	AARG and ACC mantissa
ACC+B1	=	AARG+B1	
ACC+B2	=	AARG+B2	
ACC+B3			additional ACC storage
ACC+B4			
ACC+B5			
SIGN			sign in MSB
FPFLAGS			exception flags, option bits
BEXP			BARG exponent
BARG+B0			BARG mantissa
BARG+B1			
BARG+B2			
TEMP+B0			temporary storage
TEMP+B1			

where Bx = x.

The exception flags and option bits in FPFLAGS are defined as follows:



IOV	integer overflow flag
FOV	floating point overflow flag
FUN	floating point underflow flag
FDZ	floating point divide by zero flag
RND	nearest neighbor rounding enable bit
SAT	saturate enable bit

USAGE

For the unary operations, input argument and result are in AARG. The binary operations require input arguments in AARG and BARG, and produces the result in AARG, thereby simplifying sequencing of operations.

EXCEPTION HANDLING

All routines return WREG = 0x00 upon successful completion and WREG = 0xFF, together with the appropriate flag bit set upon exception. If SAT = 0, saturation is disabled and spurious results are obtained in AARG upon an exception. If SAT = 1, saturation is enabled and only overflow or underflow exceptions produce saturated results in AARG. Saturation is not applied on a divide by zero exception.

ROUNDING

With RND = 0, rounding is disabled, and simple truncation is used, resulting in some speed enhancement. If RND = 1, nearest neighbor rounding is implemented.

INTEGER TO FLOAT CONVERSION

The routine FLOxx converts the two's complement integer in AARG to the above floating point representation, producing the result in AEXP, AARG. The routine initializes the exponent to move the radix point to the left of the MSB and then calls the normalize routine. An example is given by:

FLO24(12106) = FLO24(0x2F4A) = 0x8E3D28 = 12106.0

5

© 1994 Microchip Technology Inc.

NORMALIZE

The routine NRMxx takes an unnormalized floating point number with all bits explicit in AEXP, AARG and left shifts the mantissa and adjusts the exponent until the result has an explicit MSB = 1, followed by application of the sign bit thereby making the MSB implicit. This routine is called by FLOxx, FPAxx and FPSxx, and is usually not needed explicitly by the user since all operations producing a floating point result are implicitly normalized.

FLOAT TO INTEGER CONVERSION

The routine INTxx converts the normalized floating point number in AEXP, AARG, to a two's complement integer in AARG. After removing the bias from AEXP and precluding a result of zero or integer overflow, the mantissa in AARG is left shifted by AEXP and converted to two's complement representation. As an example, consider:

INT24(123.45) = INT24(0x87F6E6) = 0x7B = 123

ADDITION/SUBTRACTION

The floating point add routine FPAxx, takes the arguments in AEXP, AARG and BEXP, BARG and returns the sum in AEXP, AARG. If necessary, the arguments are swapped to insure that AEXP >= BEXP, and then BARG is then aligned by right shifting by AEXP - BEXP. The mantissas are then added and the result is normalized by calling NRMxx. The subtract routine FPSxx simply toggles the sign bit in BARG and calls FPAxx. Several examples are as follows:

```
FPA24( -.32212E+5,.11120E+4 )
        = FPA24( 0x8FFBA8, 0x8B0B00 )
        = 0x8FF2F8 = -.31100E+5

FPS24( .89010E+4,-.17802E+5 )
        = FPS24( 0x8E0B14, 0x8F8B14 )
        = 0x8F509E = .26703E+5
```

MULTIPLICATION

The floating point multiply routine FPMxx, takes the arguments in AEXP, AARG and BEXP, BARG and returns the product in AEXP, AARG. After testing for a zero argument, the sign and exponent of the result are computed together with testing for overflow. The mantissas are then multiplied using a standard add-shift method, followed by postnormalization if necessary. For example, consider:

```
FPM24( -.32212E+5,.11120E+4 )
        = FPM24( 0x8FFBA8, 0x8B0B00 )
        = 0x9A88A4 = -.35820E+8
```

DIVISION

The floating point divide routine FPDxx, takes the numerator in AEXP, AARG and denominator in BEXP, BARG and returns the quotient in AEXP, AARG. After testing for a zero denominator, the sign and exponent of the result are computed together with testing for dividend alignment. If the argument mantissas satisfy the inequality AARG >= BARG, the dividend AARG is right shifted by one bit and the exponent is adjusted, thereby resulting in AARG < BARG and the dividend is aligned. Alignment permits a valid division sequence and eliminates the need for postnormalization. After testing for overflow or underflow as appropriate, the mantissas are then divided using a standard shift-subtract restoring method. A simple example is given by:

```
FPD24( -.16106E+5, .24715E+5 )
        = FPD24( 0x8EFBA8, 0x8F4116 )
        = 0x80A6D4 = -.65167E+0
```


© 1994 Microchip Technology Inc.

APPENDIX A

Several algorithms for decimal to binary conversion are given below. The integer and fractional conversion algorithms are useful in both native assembly as well as high level languages, and comprise the basis for the floating point decimal to binary conversion algorithm A.3. Algorithm A.4 is a more brute force method easily implemented on a calculator or in a high-level language on a host computer.

A.1 Integer conversion algorithm[3]:

Given an integer N, where d(k) are the bit values of its n bit binary representation with d(0)=LSB,

$$N = \sum_{k=0}^{n-1} \{ d(k) * (2^k) \}$$

```
k=0
N(k) = N
while N(k) =! 0
d(k) = remainder of N(k)/2
N(k+1) = [ N(k)/2 ]
k = k + 1
endw
```

where [ ] denotes the greatest integer function.

A.2 Fractional conversion algorithm[3]:

Given a fraction N, where d(k) are the bit values of its n bit binary representation with d(1)=MSB:

$$N = \sum_{k=1}^{n} \{ d(k) * (2^{-k}) \}$$

```
k=0
N(k) = N
while k <= n
d(k) = [ N(k)*2 ]
N(k+1) = fractional part of N(k)*2
k = k + 1
endw
```

A.3 Floating point decimal to binary conversion algorithm[3]:

Given a decimal number $A = F * 10^E$, and the number of mantissa bits n, where F is not necessarily normalized, the binary floating point representation $A = f * 2^e$ can be found as follows:

normalize F and adjust E

convert 10^E to the normalized binary representation $f' * 2^e$

convert the fraction F to the binary fraction f using A.2

compute $f = f * f'$

normalize f and adjust e if necessary

The conversion of the integer 10^E can be done with algorithm A.1, or if space is available, can be performed by a table lookup. Furthermore, the conversion should be made using a number of guard bits and then rounded to an n bit result by the desired rounding method.

A.4 Floating point decimal to binary conversion algorithm:

Given a decimal number A, and the number of mantissa bits n, the bits in the mantissa of the above binary representation of A, a(k), k = 1,2,...,n, where a(1) = MSB, are given by the following algorithm:

```
z = ln A / ln 2
e = int ( z )
if e <= z
    e = e + 1
endif
x = A / (2**e)
k = 1
while k <= n
    if x >= 2**(-k)
        a(k) = 1
    else
        a(k) = 0
    endif
    x = x - a(k) * 2**(-k)
    k = k + 1
endw
```

Formally, the number A then has the floating point representation:

$$A = f * (2^e), \; f = \sum_{k=1}^{n} \{ a(k) * (2^{-k}) \}$$

5

© 1994 Microchip Technology Inc.

A simple fortran implementation of algorithm A.4 is given as follows:

```
      PROGRAM FPREP
C
      IMPLICIT NONE
C
      REAL*8 X,Y,Z
C
      INTEGER*4 K,N,P,A(32),EB,F,BIAS
C
      PARAMETER (BIAS=128)
C
1     PRINT 200
200   FORMAT(1X,'ENTER N: ',$)
      READ 100,N
100   FORMAT(I5)
C
2     PRINT 201
201   FORMAT(1X,'ENTER Y: ',$)
      READ 101,Y
101   FORMAT(E32.15)
      IF(Y.LE.0.)GOTO 1
C
      F=0
      Z=DLOG(Y)/DLOG(2.D0)
      P=INT(Z)
      IF(DBLE(P).LE.Z)P=P+1
      EB=P+BIAS
      X=Y*2.**(-P)
      DO K=1,N
          IF(X.GE.2.**(-K))THEN
              A(K)=1
          ELSE
              A(K)=0
          ENDIF
          IF(A(K).NE.0)F=IBSET(F,31-K+1)
          X=X-DBLE(A(K))*2.**(-K)
      ENDDO
C
      PRINT 300,P,(A(K),K=1,N)
300   FORMAT(1X,I5,3X,8(I2,I2,I2,I2,1X))
C
      PRINT 400,EB,F
400   FORMAT(1X,Z4,2X,Z8)
C
      GOTO 2
C
      END
```

GLOSSARY

BIASED EXPONENTS - nonnegative representation of exponents produced by adding the absolute value of the most negative exponent to a two's complement exponent.

FLOATING POINT UNDERFLOW - occurs when the real number to be represented is smaller in absolute value than the smallest floating point number.

FLOATING POINT OVERFLOW - occurs when the real number to be represented is larger in absolute value than the largest floating point number.

GUARD BITS - additional bits of precision carried in a calculation for improved rounding sensitivity.

LSB - least significant bit

MSB - most significant bit

NEAREST NEIGHBOR ROUNDING - an unbiased rounding method where a number to be rounded is rounded to its nearest neighbor in the representation, with the stipulation that if equidistant from its nearest neighbors, the neighbor with LSB equal to zero is selected.

NORMALIZATION - the process of left shifting the mantissa of an unnormalized floating point number until the MSB equals one, while decreasing the exponent by the number of left shifts.

NSB - next significant bit just to the right of the LSB.

ONE'S COMPLEMENT - a special case of the diminished radix complement for radix two systems where the value of each bit is reversed. Although sometimes used in representing positive and negative numbers, it produces two representations of the number zero.

RADIX - the base of a given number system.

RADIX POINT - separates the integer and fractional parts of a number.

SATURATION - mode of operation where integer and floating point numbers are fixed at there limiting values when an underflow or overflow is detected.

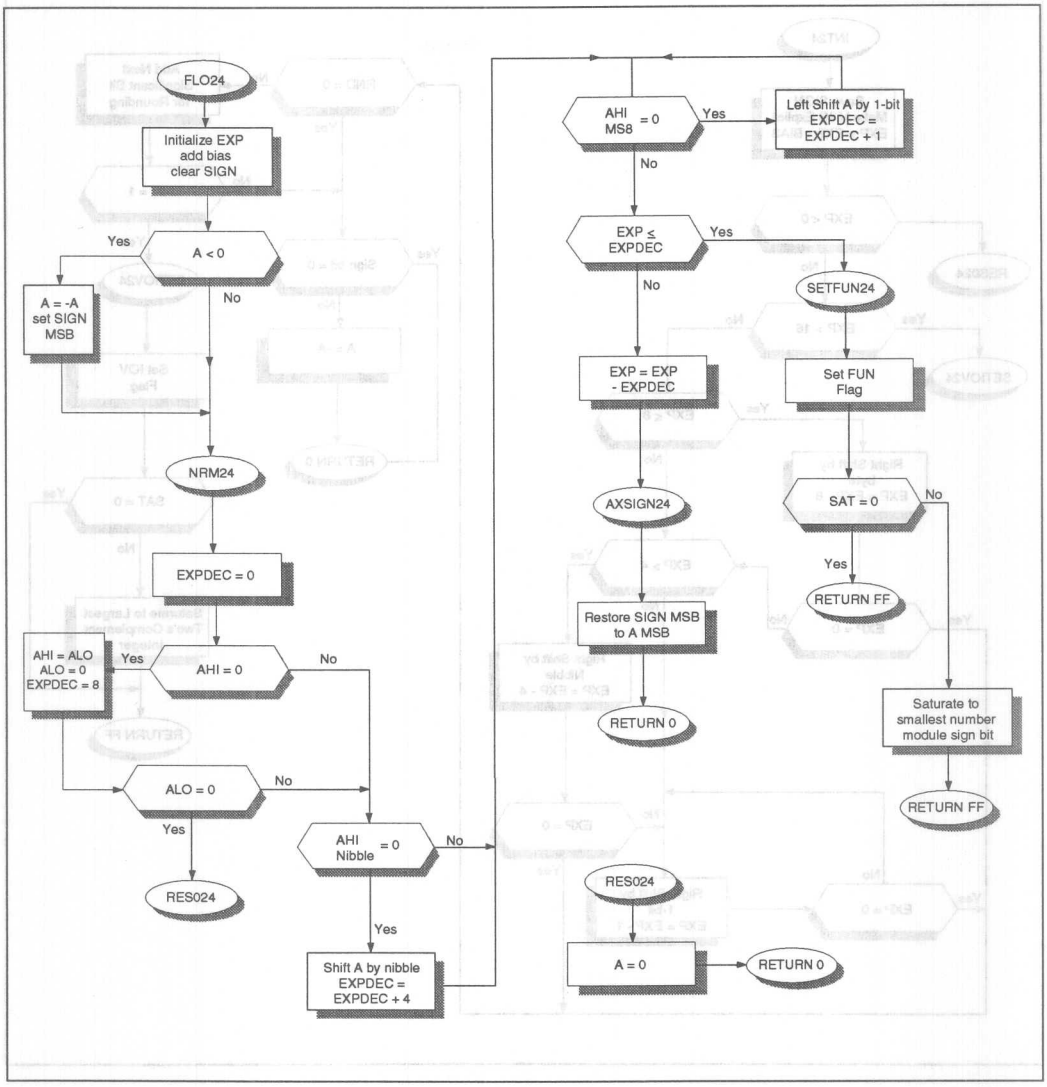
SIGN MAGNITUDE - representation of positive and negative binary numbers where the absolute value is expressed together with the appropriate value of the sign bit.

TRUNCATION - discarding any bits to the right of a given bit location.

TWO'S COMPLEMENT - a special case of radix complement for radix two systems where the value of each bit is reversed and the result is incremented by one. Producing a unique representation of zero, and covering the range -2**(n-1) to 2**(n-1)-1, this is more easily applied in addition and subtraction operations and is therefore the most commonly used method of representing positive and negative numbers.

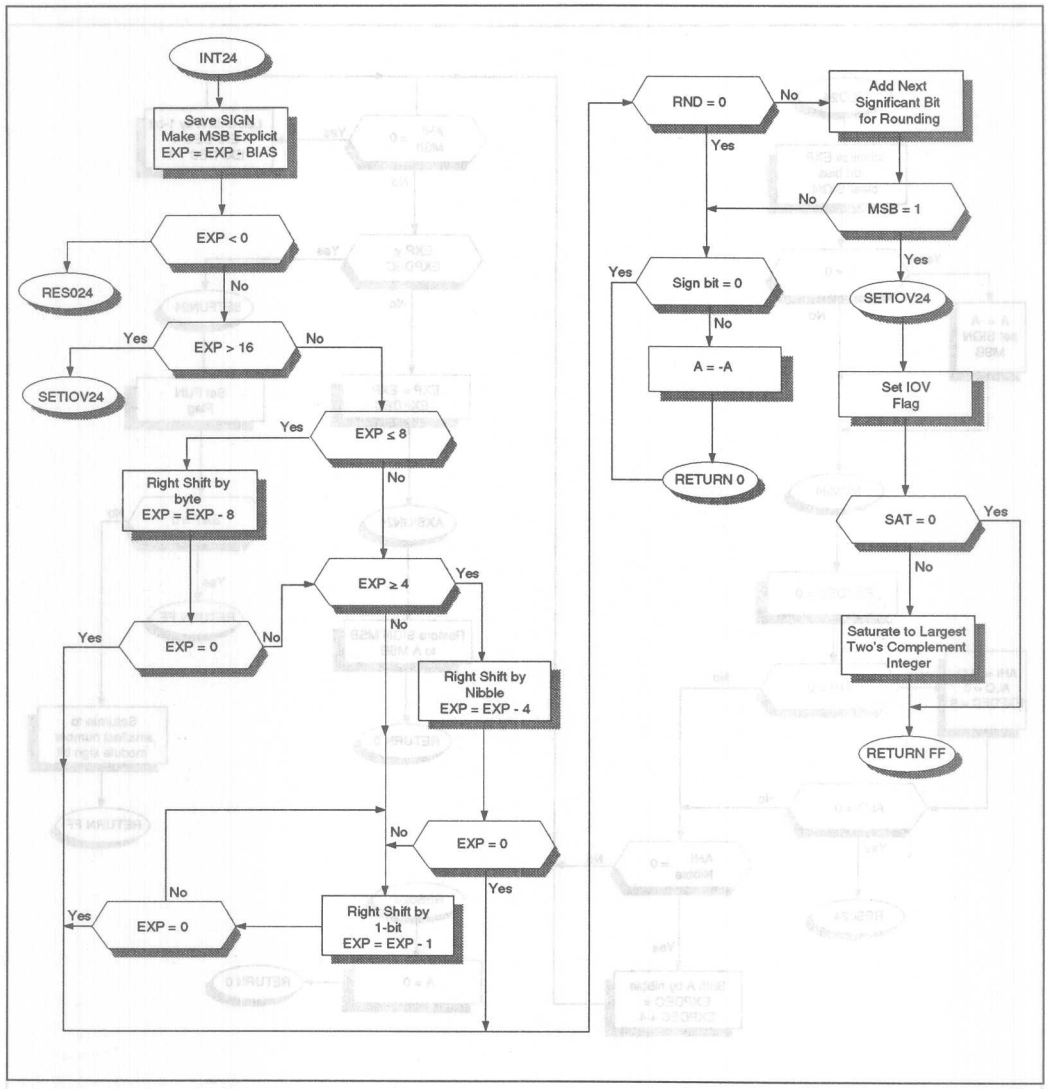
© 1994 Microchip Technology Inc.

FIGURE 1: INTEGER TO FLOAT CONVERSION



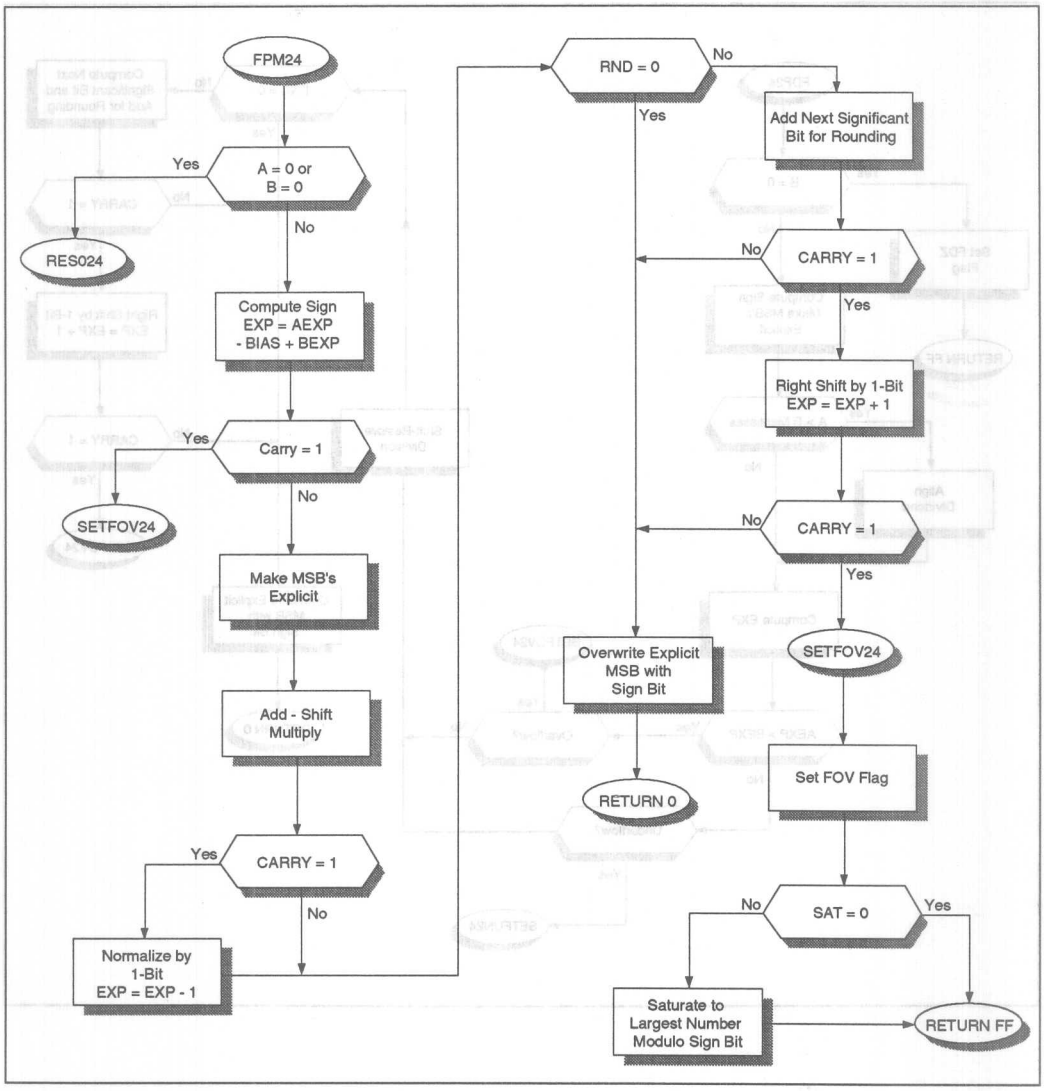
5

FIGURE 2: FLOAT TO INTEGER CONVERSION



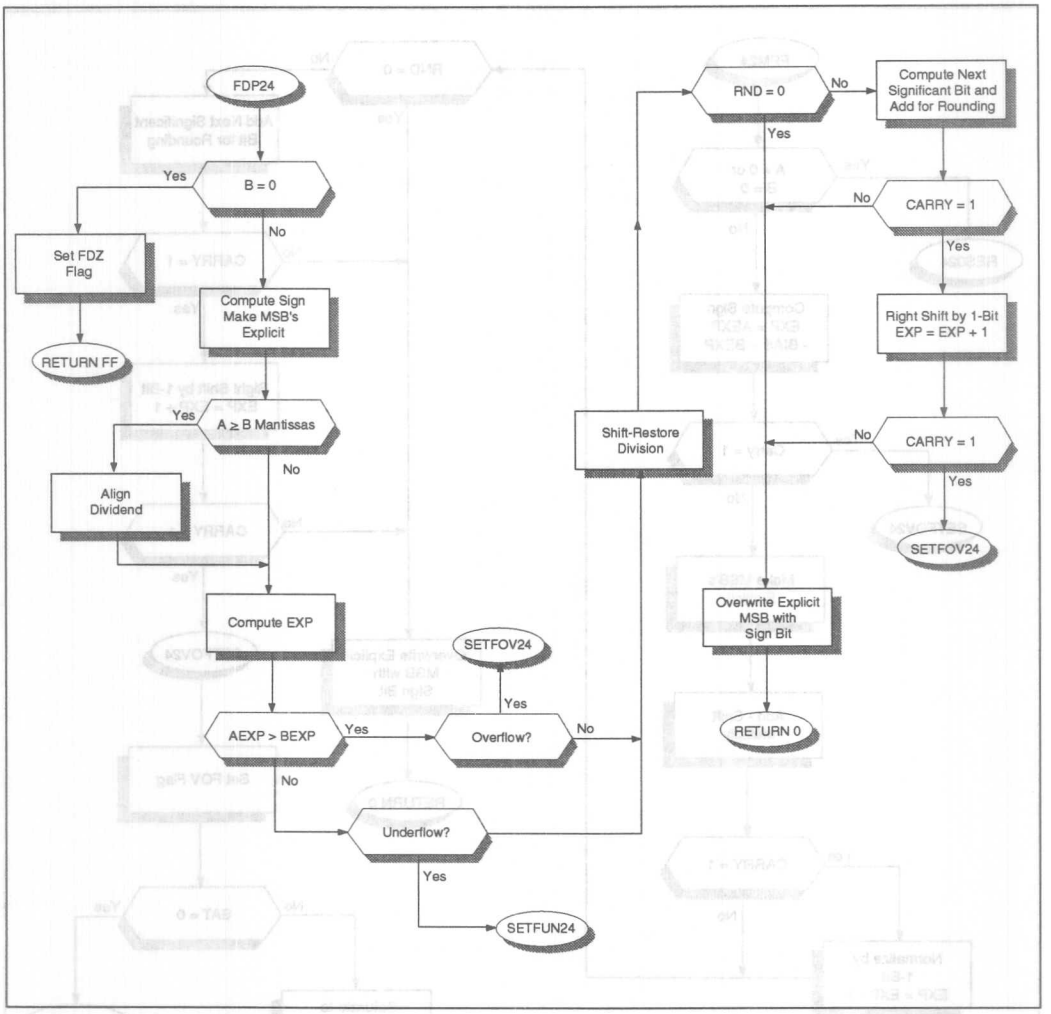
© 1994 Microchip Technology Inc.

FIGURE 3: FLOATING POINT MULTIPLY



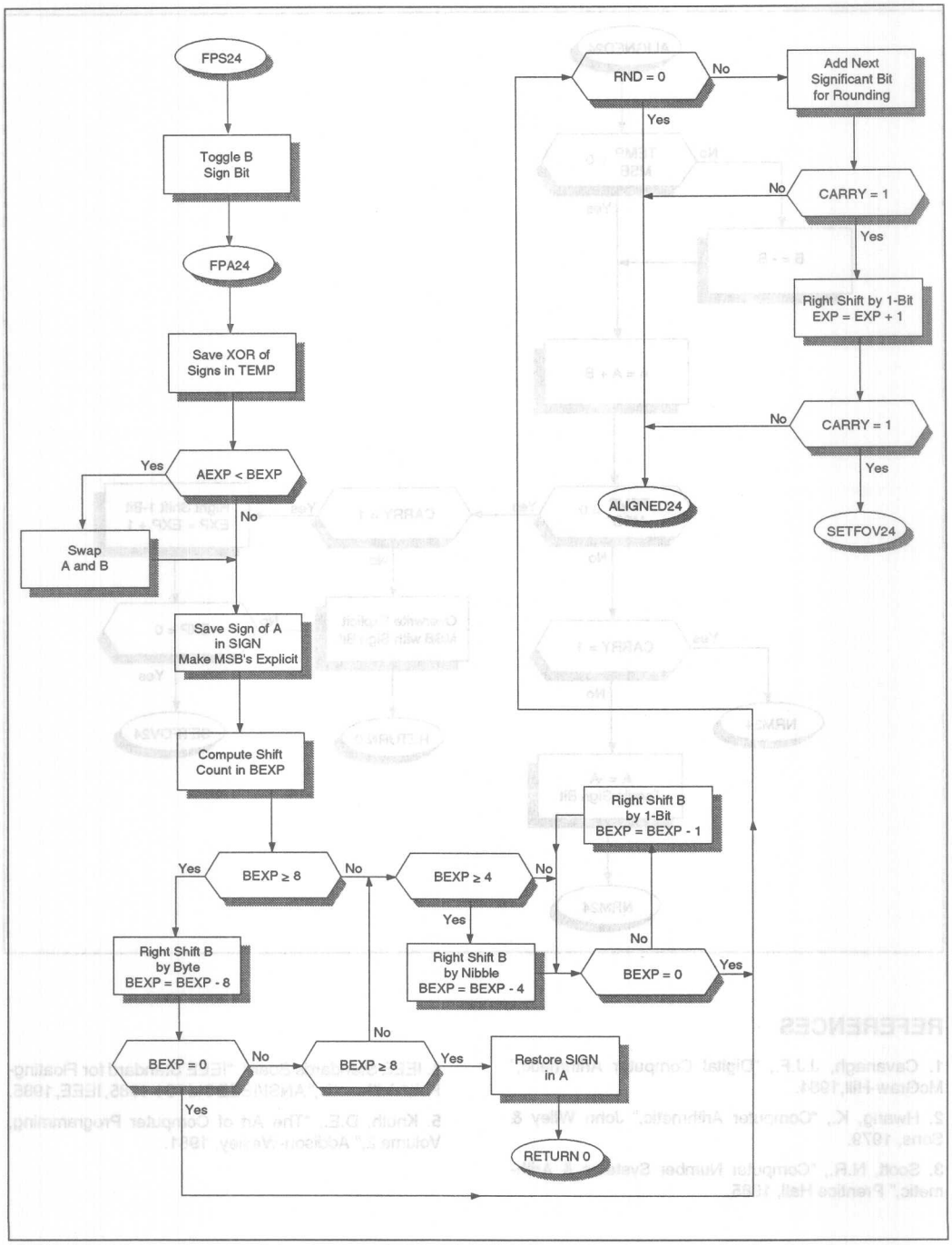
5

FIGURE 4: FLOATING POINT DIVIDE



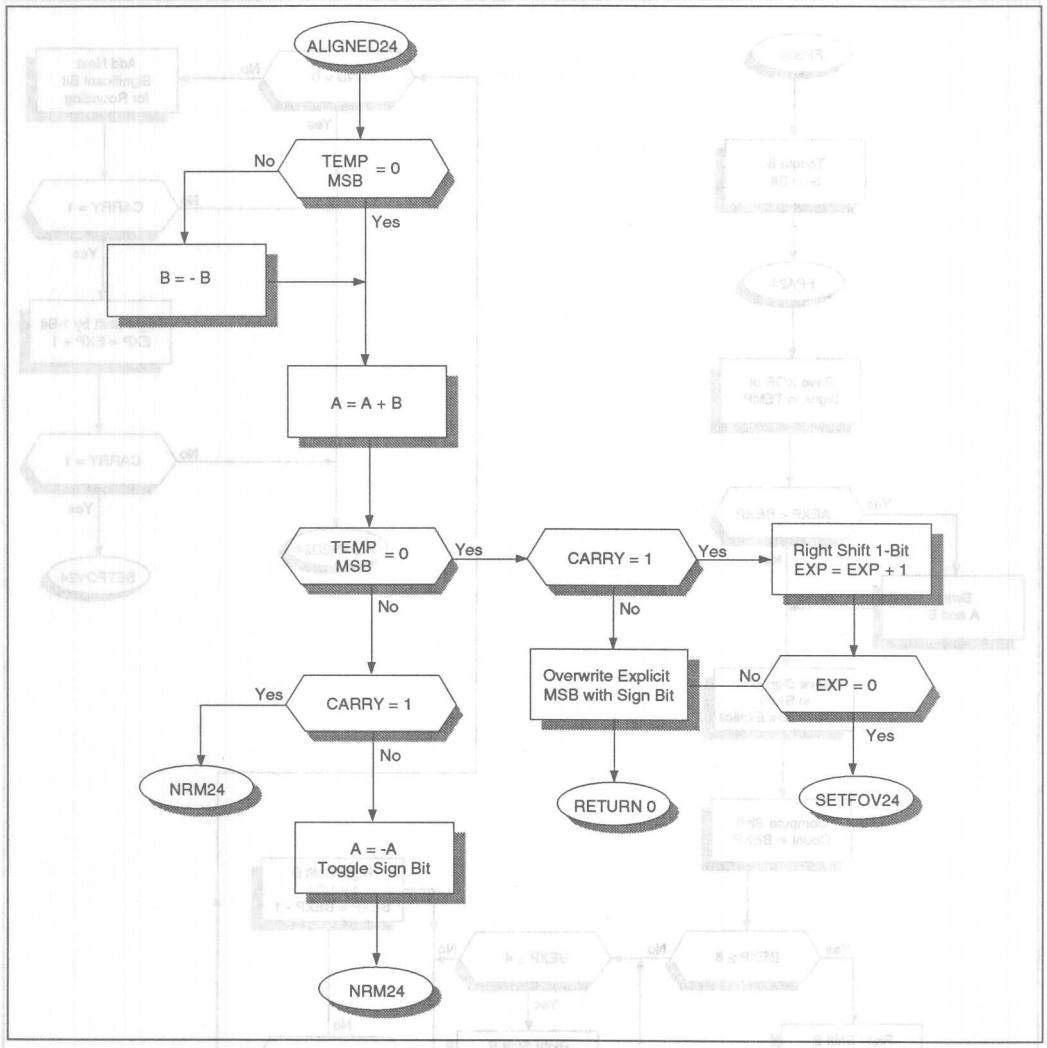
© 1994 Microchip Technology Inc.

FIGURE 5: FLOATING POINT SUBTRACT



5

FIGURE 6: NORMALIZATION



REFERENCES

1. Cavanagh, J.J.F., "Digital Computer Arithmetic," McGraw-Hill,1984.

2. Hwang, K., "Computer Arithmetic," John Wiley & Sons, 1979.

3. Scott, N.R., "Computer Number Systems & Arithmetic," Prentice Hall, 1985.

4. IEEE Standards Board, "IEEE Standard for Floating-Point Arithmetic," ANSI/IEEE Std 754-1985, IEEE, 1985.

5. Knuth, D.E., "The Art of Computer Programming, Volume 2," Addison-Wesley, 1981.

© 1994 Microchip Technology Inc.

TABLE 1: PIC17CXX FLOATING POINT PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	PM	DM	
FLO24	48 / 54	21 / 26	72	6	
INT24	57 / 65	11 / 28	101	6	
NRM24	39 / 46	15 / 20	63	6	
FPA24	116 / 131	31	212	10	
FPS24	117 / 132	32	213	10	
FPM24	175 / 189	10 / 70	224	10	
FPD24	328 / 352	11 / 277	377	11	
			1262	11	Total Memory
FLO32	60 / 67	26 / 30	120	7	
INT32	69 / 76	12 / 30	155	7	
NRM32	48 / 55	20 / 24	108	7	
FPA32	144 / 159	34	329	12	
FPS32	145 / 160	35	330	12	
FPM32	320 / 336	11 / 88	382	13	
FPD32	597 / 627	12 / 478	661	14	
			2085	14	Total Memory

TABLE 2: PIC16C5X / PIC16CXX FLOATING POINT PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	PM	DM	
FLO24	83 / 88	21 / 25	37	6	
INT24	80 / 89	14 / 31	64	6	
NRM24	72 / 77	14 / 18	26	6	
FPA24	194 / 210	44	102	11	
FPS24	196 / 212	46	104	11	
FPM24	298 / 313	11 / 218	80	11	
FPD24	469 / 495	348	117	11	
			530	11	Total Memory
FLO32	104 / 110	24 / 34	52	7	
INT32	90 / 100	15 / 32	83	6	
NRM32	90 / 96	18 / 28	38	7	
FPA32	248 / 268	50	136	14	
FPS32	250 / 270	52	138	14	
FPM32	574 / 591	12 / 315	94	14	
FPD32	929 / 968	704	152	14	
			693	14	Total Memory

© 1994 Microchip Technology Inc.

APPENDIX A

```
;       PIC16 24 BIT FLOATING POINT LIBRARY     VERSION 1.14
;       Unary operations: both input and output are in AEXP,AARG
;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG
;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.
;       All timings are worst case cycle counts
;
;         Routine               Function
;
;       FLO24           16 bit integer to 24 bit floating point conversion
;
;               Timing:             RND
;                                   0       1
;
;                       0           83      83
;                   SAT
;                       1           88      88
;
;       NRM24           normalization of unnormalized 24 bit floating point numbers
;               Timing:             RND
;                                   0       1
;
;                       0           72      72
;                   SAT
;                       1           77      77
;
;       INT24           24 bit floating point to 16 bit integer conversion
;
;               Timing:             RND
;                                   0       1
;
;                       0           80      86
;                   SAT
;                       1           80      89
;
;       FPA24           24 bit floating point add
;               Timing:             RND
;                                   0       1
;
;                       0           194     205
;                   SAT
;                       1           194     210
;
;       FPS24           24 bit floating point subtract
;               Timing:             RND
;                                   0       1
;
;                       0           196     207
;                   SAT
;                       1           196     212
;
;       FPM24           24 bit floating point multiply
;
;               Timing:             RND
```


© 1994 Microchip Technology Inc.

```
;                               0       1

;                       0       298     309
;                   SAT
;                       1       298     313

;       FPD24           24 bit floating point divide

;               Timing:             RND
;                               0       1

;                       0       469     491
;                   SAT
;                       1       469     495

                list    r=dec,x=on,t=off,p=16C71
                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       24 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = LOWBYTE = 0x00, with 0 being the only
;                       number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE        LOWBYTE

;       xxxxxxxx            .   Sxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS equ     128

;       Define library register variables

EXP             equ     0x0C    ; 8 bit biased exponent

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x0C    ; 8 bit biased exponent for argument A

AARG            equ     0x0D    ; most significant byte of mantissa for argument A

BEXP            equ     0x15    ; 8 bit biased exponent for argument B

BARG            equ     0x16    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 )  and  (EXP, ACC+B0, ACC+B1)
;               reference the same storage locations.
```

5

```
;       Define floating point library exception flags

FPFLAGS         equ     0x14    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;*****************************************************************************************
;*****************************************************************************************

;       Integer to float conversion

;       Input:  16 bit 2's complement integer right justified in AARG+B0, AARG+B1

;       Use:    CALL    FLO24

;       Output: 24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  FLOAT( AARG )

;       Max Timing:     11+72 = 83 clks         SAT = 0
;                                       11+77 = 88 clks         SAT = 1

;       Min Timing: 7+14 = 21 clks              AARG = 0
;                                       7+18 = 25 clks

;       PM: 11+26 = 37                  DM: 6

FLO24           MOVLW           16+EXPBIAS      ; initialize exponent and add bias
                MOVWF           EXP
                MOVF            ACC+B0,W
                MOVWF           SIGN
                BTFSS           ACC+B0,MSB      ; test sign
                GOTO            NRM24
                COMF            ACC+B1          ; if < 0, negate and set MSB in SIGN
                COMF            ACC+B0
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0

;-----------------------------------

;       Normalization routine

;       Input:  24 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               with sign in SIGN,MSB and other bits zero.

;       Use:    CALL    NRM24

;       Output: 24 bit normalized floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  NORMALIZE( AARG )

;       Max Timing:     10+6+7*7+7 = 72 clks            SAT = 0
;                                       10+6+7*7+1+11 = 77 clks SAT = 1

;       Min Timing: 14 clks                     AARG = 0
```


© 1994 Microchip Technology Inc.

```
;                                       5+9+4 = 18 clks

;       PM: 26                                  DM: 6

NRM24           CLRF            TEMP            ; clear exponent decrement
                MOVF            ACC+B0,W        ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM24
                MOVF            ACC+B1,W        ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                BTFSC           _Z              ; if highbyte=0, result=0
                GOTO            RES024
                CLRF            ACC+B1
                BSF             TEMP,3

NORM24          MOVF            TEMP,W
                SUBWF           EXP
                BTFSS           _Z
                BTFSS           _C
                GOTO            SETFUN24

                BCF             _C              ; clear carry bit

NORM24A         BTFSC           ACC+B0,MSB      ; if MSB=1, normalization done
                GOTO            FIXSIGN24
                RLF             ACC+B1          ; otherwise, shift left and
                RLF             ACC+B0          ; decrement EXP
                DECFSZ  EXP
                GOTO            NORM24A

                GOTO            SETFUN24        ; underflow if EXP=0

FIXSIGN24       BTFSS SIGN,MSB
                BCF             ACC+B0,MSB      ; clear explicit MSB if positive
                RETLW           0

;*****************************************************************************************
;*****************************************************************************************

;       Float to integer conversion

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Use:    CALL    INT24

;       Output: 16 bit 2's complement integer right justified in AARG+B0, AARG+B1

;       Result: AARG  <-  INT( AARG )

;       Max Timing:     26+6*6+5+13 = 80 clks           RND = 0
;                                   26+6*6+5+19 = 86 clks           RND = 1, SAT = 0
;                                   26+6*6+5+22 = 89 clks           RND = 1, SAT = 1

;       Min Timing: 9+5 = 14 clks           AARG = 0
;                                   19+5+7 = 31 clks

;       PM: 64                                  DM: 6

INT24           MOVF            ACC+B0,W        ; save sign in SIGN
                MOVWF           SIGN
                BSF             ACC+B0,MSB      ; make MSB explicit

                MOVLW           EXPBIAS         ; remove bias from EXP
                SUBWF           EXP
                BTFSS           EXP,MSB         ; if <= 0, result=0
                BTFSC           _Z
                GOTO            RES024

                MOVF            EXP,W
```

5

```
                SUBLW           16
                MOVWF           EXP
                BTFSS           _Z
                BTFSC           EXP,MSB
                GOTO            SETIOV24

                MOVLW           8                   ; do byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT24
                MOVWF           EXP
                RLF             ACC+B1              ; rotate next bit for rounding
                MOVF            ACC+B0,W
                MOVWF           ACC+B1
                CLRF            ACC+B0

                MOVF            EXP,W               ; shift completed if EXP = 0
                BTFSC           _Z
                GOTO            SHIFT24OK

SHIFT24         BCF             _C
                RRF             ACC+B0              ; right shift by EXP
                RRF             ACC+B1
                DECFSZ  EXP
                GOTO            SHIFT24

SHIFT24OK       BTFSC FPFLAGS,RND
                BTFSS           ACC+B1,LSB
                GOTO            INT24OK
                CLRF            ACC+B2
                RLF             ACC+B2,W
                ADDWF           ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                BTFSC           ACC+B0,MSB          ; test for overflow
                GOTO            SETIOV24

INT24OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                RETLW           0

RES024          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV24        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two's
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                MOVLW           0xFF
                MOVWF           ACC+B0              ; SIGN = 0, 0x 7F FF
                MOVWF           ACC+B1              ; SIGN = 1, 0x 80 00
                RLF             SIGN
                RRF             ACC+B0
                RETLW           0xFF                ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Multiply
```


© 1994 Microchip Technology Inc.

```
;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPM24

;       Output: 24 bit floating point product in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG * BARG

;       Max Timing:     25+15*16+15+18 = 298 clks       RND = 0
;                                       25+15*16+15+29 = 309 clks       RND = 1, SAT = 0
;                                       25+15*16+15+33 = 313 clks       RND = 1, SAT = 1

;       Min Timing:     6+5 = 11 clks           AARG * BARG = 0
;                                       24+15*11+14+15 = 218 clks

;       PM: 80                                  DM: 11

FPM24           MOVF            AEXP,W          ; test for zero arguments
                BTFSS           _Z
                MOVF            BEXP,W
                BTFSC           _Z
                GOTO            RES024

M24BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN            ; save sign in SIGN

                MOVF            BEXP,W
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN24
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV24        ; set multiply overflow flag
                GOTO            MOK24

MTUN24          ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN24

MOK24           BSF             AARG+B0,MSB     ; make argument MSB's explicit
                BSF             BARG+B0,MSB
                BCF             _C
                CLRF            ACC+B2          ; clear initial partial product
                CLRF            ACC+B3
                MOVLW           16
                MOVWF           TEMP            ; initialize counter

MLOOP24         BTFSS           AARG+B1,LSB     ; test high byte
                GOTO            MNOADD24

MADD24          MOVF            BARG+B1,W
                ADDWF           ACC+B3
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ  BARG+B0,W
                ADDWF           ACC+B2

MNOADD24        RRF       ACC+B2
                RRF             ACC+B3
                RRF             ACC+B0
                RRF             ACC+B1
                BCF             _C
                DECFSZ  TEMP
                GOTO            MLOOP24
```

5

```
                BTFSC           ACC+B2,MSB          ; check for postnormalization
                GOTO            MROUND24
                RLF             ACC+B0
                RLF             ACC+B3
                RLF             ACC+B2
                DECF            EXP

MROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            MUL24OK
                RLF             ACC+B0              ; rotate next significant bit into
                BTFSC           _C
                INCF            ACC+B3              ; carry for rounding
                BTFSC           _Z
                INCF            ACC+B2

                BTFSS           _Z                  ; has rounding caused carryout?
                GOTO            MUL24OK
                RRF             ACC+B2              ; if so, right shift
                RRF             ACC+B3
                INCF            EXP
                BTFSC           _C                  ; check for overflow
                GOTO            SETFOV24

MUL24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B2,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B3,W
                MOVWF           AARG+B1
                RETLW           0

SETFOV24        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0xFF
                MOVWF           AEXP                ; saturate to largest floating
                MOVWF           AARG+B0             ; point number = 0x FF 7F FF
                MOVWF           AARG+B1             ; modulo the appropriate sign bit
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Divide

;       Input:  24 bit floating point dividend in AEXP, AARG+B0, AARG+B1
;               24 bit floating point divisor in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPD24

;       Output: 24 bit floating point quotient in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG / BARG

;       Max Timing:     29+13+15*26+25+12 = 469 clks    RND = 0
;                               29+13+15*26+25+34 = 491 clks    RND = 1, SAT = 0
;                               29+13+15*26+25+38 = 495 clks    RND = 1, SAT = 1

;       Min Timing:     24+12+15*20+12 = 348 clks

;       PM: 117                             DM: 11

FPD24           MOVF            BEXP,W              ; test for divide by zero
```


© 1994 Microchip Technology Inc.

```
                BTFSC           _Z
                GOTO            SETFDZ24

D24BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN24        CLRF            TEMP                    ; clear align increment
                MOVF            AARG+B0,W
                MOVWF           ACC+B2                  ; test for alignment
                MOVF            AARG+B1,W
                MOVWF           ACC+B3

                MOVF            BARG+B1,W
                SUBWF           ACC+B3
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B2

                CLRF            ACC+B2
                CLRF            ACC+B3
                BTFSS           _C
                GOTO            DALIGN24OK

                BCF             _C                      ; align if necessary
                RRF             ACC+B0
                RRF             ACC+B1
                RRF             ACC+B2
                MOVLW           0x01
                MOVWF           TEMP                    ; save align increment

DALIGN24OK      MOVF            BEXP,W                  ; compare AEXP and BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB24

AGEB24          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            DARGOK24                ; set overflow flag

ALTB24          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN24                ; set underflow flag

DARGOK24        MOVLW           16                      ; initialize counter
                MOVWF           TEMP+B1

DLOOP24         RLF             ACC+B3                  ; left shift
                RLF             ACC+B2
                RLF             ACC+B1
                RLF             ACC+B0
                RLF             TEMP
                MOVF            BARG+B1,W               ; subtract
                SUBWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B0
```


© 1994 Microchip Technology Inc.

```
                RLF             BARG+B0,W
                IORWF           TEMP

                BTFSS           TEMP,LSB            ; test for restore
                GOTO            DREST24

                BSF             ACC+B3,LSB
                GOTO            DOK24

DREST24         MOVF            BARG+B1,W           ; restore if necessary
                ADDWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           ACC+B0
                BCF             ACC+B3,LSB

DOK24           DECFSZ          TEMP+B1
                GOTO            DLOOP24

DROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            DIV24OK
                BCF             _C
                RLF             ACC+B1              ; compute next significant bit
                RLF             ACC+B0              ; for rounding
                RLF             TEMP

                MOVF            BARG+B1,W           ; subtract
                SUBWF           ACC+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           ACC+B0

                RLF             BARG+B0,W
                IORWF           TEMP,W
                ANDLW           0x01

                ADDWF           ACC+B3
                BTFSC           _C
                INCF            ACC+B2

                BTFSS           _Z                  ; test if rounding caused carryout
                GOTO            DIV24OK
                RRF             ACC+B2
                RRF             ACC+B3
                INCF            EXP
                BTFSC           _Z                  ; test for overflow
                GOTO            SETFOV24

DIV24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVF            ACC+B2,W
                MOVWF           AARG+B0             ; move result to AARG
                MOVF            ACC+B3,W
                MOVWF           AARG+B1

                RETLW           0

SETFUN24        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVWF           AEXP                ; point number = 0x 01 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
```


© 1994 Microchip Technology Inc.

```
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                    ; return error code in WREG

SETFDZ24        BSF             FPFLAGS,FDZ             ; set divide by zero flag
                RETLW           0xFF

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Subtract

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPS24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     2+194 = 196 clks                RND = 0
;                                       2+205 = 207 clks        RND = 1, SAT = 0
;                                       2+210 = 212 clks        RND = 1, SAT = 1

;       Min Timing:     2+44 = 46 clks

;       PM: 2+102 = 104         DM: 11

FPS24           MOVLW           0x80
                XORWF           BARG+B0

;-----------------------------------------

;       Floating Point Add

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPA24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     25+25+6*6+5+31+72 = 194 clks    RND = 0
;                                       25+25+6*6+5+42+72 = 205 clks    RND = 1, SAT = 0
;                                       25+25+6*6+5+42+77 = 210 clks    RND = 1, SAT = 1

;       Min Timing:     22+18+4 = 44 clks

;       PM: 102                                 DM: 11

FPA24           MOVF            AARG+B0,W               ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVWF           TEMP

                MOVF            AEXP,W                  ; use AARG if AEXP >= BEXP
                SUBWF           BEXP,W
                BTFSS           _C
                GOTO            USEA24

                MOVF            BEXP,W
                MOVWF           ACC+B2                  ; otherwise, swap AARG and BARG
                MOVF            AEXP,W
                MOVWF           BEXP
                MOVF            ACC+B2,W
                MOVWF           AEXP
```

5

© 1994 Microchip Technology Inc.

```
                MOVF    BARG+B0,W
                MOVWF   ACC+B2
                MOVF    AARG+B0,W
                MOVWF   BARG+B0
                MOVF    ACC+B2,W
                MOVWF   AARG+B0

                MOVF    BARG+B1,W
                MOVWF   ACC+B2
                MOVF    AARG+B1,W
                MOVWF   BARG+B1
                MOVF    ACC+B2,W
                MOVWF   AARG+B1

USEA24          MOVF    AARG+B0,W
                MOVWF   SIGN            ; save sign in SIGN
                BSF     AARG+B0,MSB     ; make MSB's explicit
                BSF     BARG+B0,MSB

                MOVF    BARG+B0,W
                MOVWF   ACC+B2
                MOVF    BARG+B1,W
                MOVWF   ACC+B3

                MOVF    BEXP,W          ; compute shift count in BEXP
                SUBWF   AEXP,W
                MOVWF   BEXP
                BTFSC   _Z
                GOTO    AROUND24

                MOVLW   8
                SUBWF   BEXP,W
                BTFSS   _C              ; if BEXP >= 8, do byte shift
                GOTO    ALIGNB24
                MOVWF   BEXP
                RLF     ACC+B3          ; rotate next bit for rounding
                MOVF    ACC+B2,W
                MOVWF   ACC+B3
                CLRF    ACC+B2

ALIGNB24        MOVF    BEXP,W          ; already aligned if BEXP = 0
                BTFSC   _Z
                GOTO    AROUND24

ALOOPB24        BCF     _C              ; right shift by BEXP
                RRF     ACC+B2
                RRF     ACC+B3
                DECFSZ  BEXP
                GOTO    ALOOPB24

AROUND24        BTFSC   FPFLAGS,RND
                BTFSS   ACC+B3,LSB
                GOTO    ALIGNED24
                BTFSS   _C
                GOTO    ALIGNED24
                INCF    ACC+B3          ; carry for rounding
                BTFSC   _Z
                INCF    ACC+B2
                BTFSS   _Z
                GOTO    ALIGNED24
                RRF     ACC+B2
                RRF     ACC+B3
                INCF    EXP
                BTFSC   _Z
                GOTO    SETFOV24

ALIGNED24       BTFSS   TEMP,MSB        ; negate if signs opposite
                GOTO    AOK24
                COMF    ACC+B3
```


© 1994 Microchip Technology Inc.

```
                COMF            ACC+B2
                INCF            ACC+B3
                BTFSC           _Z
                INCF            ACC+B2

AOK24           MOVF            ACC+B3,W            ; add
                ADDWF           AARG+B1
                BTFSC           _C
                INCF            ACC+B0
                MOVF            ACC+B2,W
                ADDWF           AARG+B0
                BTFSC           TEMP,MSB
                GOTO            ACOMP24
                BTFSS           _C
                GOTO            FIXSIGN24

                RRF             AARG+B0             ; shift right and increment EXP
                RRF             AARG+B1
                INCFSZ  AEXP
                GOTO            FIXSIGN24
                GOTO            SETFOV24

ACOMP24         BTFSC           _C
                GOTO            NRM24               ; normalize and fix sign
                COMF            AARG+B1             ; negate, toggle sign bit and
                COMF            AARG+B0             ; then normalize
                INCF            AARG+B1
                BTFSC           _Z
                INCF            AARG+B0

                MOVLW           0x80
                XORWF           SIGN
                GOTO            NRM24

                END
```

5

APPENDIX B

```
;       PIC17 24 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Max timings are worst case cycle counts, while Min timings are non-exception
;       best case cycle counts.

;         Routine               Function

;       FLO24           16 bit integer to 24 bit floating point conversion

;       Max Timing:                 RND
;                               0       1

;                       0       48      48
;                SAT
;                       1       55      55

;       NRM24           normalization of unnormalized 24 bit floating point numbers

;       Max Timing:                 RND
;                               0       1

;                       0       39      39
;                SAT
;                       1       46      46


;       INT24           24 bit floating point to 16 bit integer conversion


;       Max Timing:                 RND
;                               0       1

;                       0       57      61
;                SAT
;                       1       57      65

;       FPA24           24 bit floating point add

;       Max Timing:                 RND
;                               0       1

;                       0       116     131
;                SAT
;                       1       116     131

;       FPS24           24 bit floating point subtract

;       Max Timing:                 RND
;                               0       1

;                       0       117     132
;                SAT
;                       1       117     132

;       FPM24           24 bit floating point multiply

;       Max Timing:                 RND
;                               0       1

;                       0       175     184
```


© 1994 Microchip Technology Inc.

```
;                   SAT
;                        1        175      189

;       FPD24            24 bit floating point divide

;       Max Timing:                 RND
;                                   0        1

;                        0        328      347
;                   SAT
;                        1        328      352


                list    r=dec,x=on,t=off,p=17C42
                include <PIC17.INC>

;*****************************************************************************************
;*****************************************************************************************

;       24 bit floating point representation

;       EXPONENT         8 bit biased exponent
;                        It is important to note that the use of biased exponents produces
;                        a unique representation of a floating point 0, given by
;                        EXP = HIGHBYTE = LOWBYTE = 0x00, with 0 being the only
;                        number with EXP = 0.

;       HIGHBYTE         8 bit most significant byte of sign-magnitude representation, with
;                        SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                        left of MSB

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                        RADIX
;       EXPONENT         POINT   HIGHBYTE        LOWBYTE

;       xxxxxxxx            .    Sxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS         equ     128
;       Define library register variables

EXP             equ     0x18    ; 8 bit biased exponent

ACC             equ     0x19    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x1F    ; save location for sign in MSB

TEMP            equ     0x25    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x18    ; 8 bit biased exponent for argument A

AARG            equ     0x19    ; most significant byte of mantissa for argument A

BEXP            equ     0x21    ; 8 bit biased exponent for argument B

BARG            equ     0x22    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 ) and (EXP, ACC+B0, ACC+B1)
;               reference the same storage locations.

;       Define floating point library exception flags
```

5

© 1994 Microchip Technology Inc.

```
FPFLAGS         equ     0x20    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = unbiased rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;*****************************************************************************************
;*****************************************************************************************

;       Integer to float conversion

;       Input:  16 bit 2's complement integer right justified in AARG+B0, AARG+B1

;       Use:    CALL    FLO24

;       Output: 24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  FLOAT( AARG )

;       Max Timing:     9+39 = 48 clks          SAT = 0
;                                   9+45 = 54 clks          SAT = 1

;       Min Timing:     6+15 = 21 clks          AARG = 0
;                                   6+20 = 26 clks

;       PM: 9+63 = 72                   DM: 6

FLO24           MOVLW           16+EXPBIAS          ; initialize exponent and add bias
                MOVWF           EXP
                MOVFP           ACC+B0,SIGN         ; save sign in SIGN
                BTFSS           ACC+B0,MSB          ; test sign
                GOTO            NRM24
                COMF            ACC+B1              ; if < 0, negate, set MSB in SIGN
                COMF            ACC+B0
                INFSNZ  ACC+B1
                INCF            ACC+B0

;-----------------------------------------------------------------------------------------

;       Normalization routine

;       Input:  24 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               with sign in SIGN,MSB.

;       Use:    CALL    NRM24

;       Output: 24 bit normalized floating point number in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  NORMALIZE( AARG )

;       Max Timing:     3+12+16+8 = 39 clks             SAT = 0
;                                   3+12+16+15 = 46 clks        SAT = 1

;       Min Timing:     10+5 = 15 clks          AARG = 0
;                                   3+5+4+8 = 20 clks

;       PM: 63                          DM: 6
```


© 1994 Microchip Technology Inc.

```
NRM24           CLRF    TEMP,W          ; clear exponent decrement
                CPFSGT  ACC+B0          ; test if highbyte=0
                GOTO    NRM24A

TNIB24          MOVLW   0xF0            ; test if highnibble=0
                ANDWF   ACC+B0,W
                TSTFSZ  WREG
                GOTO    NORM24
                SWAPF   ACC+B0          ; if so, shift 4 bits
                SWAPF   ACC+B1,W
                ANDLW   0x0F
                ADDWF   ACC+B0
                SWAPF   ACC+B1,W
                ANDLW   0xF0
                MOVPF   WREG,ACC+B1

                BSF     TEMP,2          ; increase decrement by 4

NORM24          BCF     _C              ; clear carry bit

                BTFSC   ACC+B0,MSB      ; if MSB=1, normalization done
                GOTO    TNORMUN24
                RLCF    ACC+B1          ; otherwise, shift left and
                RLCF    ACC+B0          ; increment decrement
                INCF    TEMP
                BTFSC   ACC+B0,MSB
                GOTO    TNORMUN24
                RLCF    ACC+B1
                RLCF    ACC+B0
                INCF    TEMP
                BTFSC   ACC+B0,MSB      ; since highnibble != 0, at most
                GOTO    TNORMUN24       ; 3 left shifts are required
                RLCF    ACC+B1
                RLCF    ACC+B0
                INCF    TEMP

TNORMUN24       MOVFP   TEMP,WREG       ; if EXP <= decrement in TEMP,
                CPFSGT  EXP             ; floating point underflow has
                GOTO    SETFUN24        ; occured
                SUBWF   EXP             ; otherwise, compute EXP

FIXSIGN24       BTFSS   SIGN,MSB
                BCF     ACC+B0,MSB      ; clear explicit MSB if positive
                RETLW   0

NRM24A          MOVFP   ACC+B1,ACC+B0   ; if so, shift 8 bits by move
                CLRF    ACC+B1
                BSF     TEMP,3          ; increase decrement by 8
                CPFSGT  ACC+B0          ; if highbyte=0, result=0
                GOTO    RES024

                MOVLW   0xF0            ; test if highnibble=0
                ANDWF   ACC+B0,W
                TSTFSZ  WREG
                GOTO    NORM24A
                SWAPF   ACC+B0          ; if so, shift 4 bits

                BSF     TEMP,2          ; increase decrement by 4

NORM24A         BCF     _C              ; clear carry bit

                BTFSC   ACC+B0,MSB      ; if MSB=1, normalization done
                GOTO    TNORMUN24
                RLCF    ACC+B0          ; otherwise, shift left and
                INCF    TEMP            ; increment decrement
                BTFSC   ACC+B0,MSB
                GOTO    TNORMUN24
                RLCF    ACC+B0
                INCF    TEMP
```

5

© 1994 Microchip Technology Inc.

```
                BTFSC   ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO    TNORMUN24           ; 3 left shifts are required
                RLCF    ACC+B0
                INCF    TEMP
                GOTO    TNORMUN24

;*****************************************************************************************
;*****************************************************************************************

;       Float to integer conversion

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1

;       Use:    CALL    INT24

;       Output: 16 bit 2's complement integer right justified in AARG+B0, AARG+B1

;       Result: AARG  <-  INT( AARG )

;       Max Timing:     11+34+12 = 57 clks              RND = 0
;                                       11+34+16 = 61 clks      RND = 1, SAT = 0
;                                       11+34+20 = 65 clks      RND = 1, SAT = 1

;       Min Timing:     6+5 = 11 clks           AARG = 0
;                                       11+10+7 = 28 clks

;       PM: 101                                 DM: 6

INT24           MOVFP   ACC+B0,SIGN         ; save sign in SIGN
                BSF     ACC+B0,MSB          ; make MSB explicit
                MOVLW   EXPBIAS             ; remove bias from EXP
                CPFSGT  EXP                 ; if <= 0, result=0
                GOTO    RES024
                SUBWF   EXP
                MOVLW   16
                CPFSLT  EXP                 ; if >= 16, integer overflow
                GOTO    SETIOV24            ; will occur
                SUBWF   EXP,W
                NEGW    EXP
                MOVLW   7                   ; do byte shift if EXP >= 8
                CPFSGT  EXP
                GOTO    SNIB24
                SUBWF   EXP                 ; EXP = EXP - 7
                RLCF    ACC+B1              ; rotate next bit for rounding
                MOVFP   ACC+B0,ACC+B1
                CLRF    ACC+B0
                DCFSNZ  EXP                 ; EXP = EXP - 1
                GOTO    SHIFT24OK           ; shift completed if EXP = 0

SNIB24A         MOVLW   3                   ; do nibble shift if EXP >= 4
                CPFSGT  EXP
                GOTO    SHIFT24A
                SUBWF   EXP                 ; EXP = EXP - 3
                SWAPF   ACC+B1,W
                MOVFP   WREG,ACC+B2         ; save for rounding
                ANDLW   0x0F
                MOVPF   WREG,ACC+B1
                RLCF    ACC+B2              ; rotate next bit for rounding
                DCFSNZ  EXP                 ; EXP = EXP - 1
                GOTO    SHIFT24OK           ; shift completed if EXP = 0

SHIFT24A        BCF     _C                  ; at most 3 right shifts are required
                RRCF    ACC+B1              ; right shift by EXP
                DCFSNZ  EXP
                GOTO    SHIFT24OK           ; shift completed if EXP = 0
```


© 1994 Microchip Technology Inc.

```
                BCF             _C
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                GOTO            SHIFT24OK

SNIB24          MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT24
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B1,W
                MOVFP           WREG,ACC+B2         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1
                SWAPF           ACC+B0,W
                ANDLW           0xF0
                ADDWF           ACC+B1
                SWAPF           ACC+B0,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B0
                RLCF            ACC+B2              ; rotate next bit for rounding
                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT24OK           ; shift completed if EXP = 0

SHIFT24         BCF             _C                  ; at most 3 right shifts are required
                RRCF            ACC+B0              ; right shift by EXP
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                DCFSNZ          EXP
                GOTO            SHIFT24OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1

SHIFT24OK       BTFSC           FPFLAGS,RND
                BTFSS           ACC+B1,LSB
                GOTO            INT24OK
                CLRF            WREG
                ADDWFC          ACC+B1              ; add next bit for rounding
                ADDWFC          ACC+B0
                BTFSC           ACC+B0,MSB
                GOTO            SETIOV24

INT24OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B1
                COMF            ACC+B0
                INFSNZ          ACC+B1
                INCF            ACC+B0
                RETLW           0

RES024          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV24        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two's
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                SETF            ACC+B0              ; SIGN = 0, 0x 7F FF
```

5

© 1994 Microchip Technology Inc.

```
                MOVPF           ACC+B0,ACC+B1   ; SIGN = 1, 0x 80 00
                RLCF            SIGN
                RRCF            ACC+B0
                RETLW           0xFF            ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Multiply

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPM24

;       Output: 24 bit floating point product in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG * BARG

;       Max Timing:     19+140+16 = 175 clks            RND = 0
;                                       19+140+25 = 184 clks        RND = 1, SAT = 0
;                                       19+140+30 = 189 clks        RND = 1, SAT = 1

;       Min Timing:     5+5 = 10 clks       AARG * BARG = 0
;                                       13+5+37+15 = 70 clks

;       PM: 21+169+34 = 224             DM: 10

MUL16           macro

;       Max Timing:     10+14*9+4 = 140 clks
;       Min Timing:     33+4 = 37 clks


;       PM: 3+16*2+4+14*9+4 = 169       DM: 6
                variable i = 0
                variable j = 1

                BCF             _C                  ; clear carry bit

                CLRF            ACC+B2              ; clear initial partial product
                CLRF            ACC+B3

                while   i < 16

                if      i < 8

                BTFSC           AARG+B1,i           ; test low byte

                else

                BTFSC           AARG+B0,i-8         ; test high byte

                endif

                GOTO            Madd24#v( i )

                i += 1

                endw

;       we cannot get here since MSB's are explicit

Madd240         MOVFP           BARG+B0,ACC+B2 ; if we get here, ACC=0 so move
                MOVFP           BARG+B1,ACC+B3  ; BARG for add

                RRCF            ACC+B2
                RRCF            ACC+B3
```


© 1994 Microchip Technology Inc.

```
                while j < 15

                if j < 8
                BTFSS           AARG+B1,j           ; test low byte
                else

                BTFSS           AARG+B0,j-8         ; test high byte

                endif

                GOTO            Mnoadd24#v( j )

Madd24#v( j )   MOVFP           BARG+B1,WREG
                ADDWF           ACC+B3              ; add lsb
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B2              ; add msb

Mnoadd24#v( j ) RRCF            ACC+B2
                RRCF            ACC+B3

                if j > 13

                RRCF            ACC+B1              ; save bits for rounding

                endif

                if j < 14

                BCF             _C                  ; clear carry when rotate possible

                endif

                j += 1

                endw

Madd2415        MOVFP           BARG+B1,WREG
                ADDWF           ACC+B3              ; add lsb
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B2              ; add msb

                endm

FPM24           CLRF            WREG                ; test for zero arguments
                CPFSEQ          BEXP
                CPFSGT          AEXP
                GOTO            RES024

M24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN

                MOVFP           BEXP,WREG
                ADDWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            MTUN24

                ADDWF           EXP                 ; remove bias and overflow test
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            MOK24

MTUN24          ADDWF           EXP                 ; remove bias and underflow test
                BTFSS           _C
```

5

```
                GOTO            SETFUN24

MOK24           BSF             AARG+B0,MSB         ; make argument MSB's explicit
                BSF             BARG+B0,MSB

                MUL16                               ; use macro for multiplication

                BTFSC           _C                  ; check for postnormalization
                GOTO            MRIGHT24
                DECF            EXP
                GOTO            MROUND24
MRIGHT24        RRCF            ACC+B2
                RRCF            ACC+B3
                RRCF            ACC+B1

MROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            MUL24OK
                CLRF            WREG
                RLCF            ACC+B1              ; rotate next significant bit into
                ADDWFC          ACC+B3              ; carry for rounding
                ADDWFC          ACC+B2

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            MUL24OK
                RRCF            ACC+B2              ; if so, right shift
                RRCF            ACC+B3
                INFSNZ          EXP                 ; test for floating point overflow
                GOTO            SETFOV24

MUL24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVFP           ACC+B2,AARG+B0      ; move result to AARG
                MOVFP           ACC+B3,AARG+B1
                RETLW           0

SETFOV24        BSF             FPFLAGS,FOV         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                SETF            AEXP                ; saturate to largest floating
                SETF            AARG+B0             ; point number = 0x FF 7F FF
                SETF            AARG+B1             ; modulo the appropriate sign bit
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Divide

;       Input:  24 bit floating point dividend in AEXP, AARG+B0, AARG+B1
;               24 bit floating point divisor in BEXP, BARG+B0, BARG+B1

;       Use:    CALL    FPD24

;       Output: 24 bit floating point quotient in AEXP, AARG+B0, AARG+B1

;       Result: AARG  <-  AARG / BARG

;       Max Timing:     22+11+285+10 = 328 clks RND = 0
;                                      22+11+285+29 = 347 clks RND = 1, SAT = 0
;                                      22+11+285+34 = 352 clks RND = 1, SAT = 1

;       Min Timing: 6+5 = 11 clks              AARG = 0
;                                      30+237+10 = 277 clks
```


© 1994 Microchip Technology Inc.

```
;       PM: 36+301+40 = 377                     DM: 11

DIV16           macro

;       Timing: 1+16*20 = 321 clks

                variable i = 0

                BCF             _C

                while i < 16

                CLRF            TEMP            ; left shift
                RLCF            ACC+B3
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP

                MOVFP           BARG+B1,WREG    ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                CLRF            WREG
                SUBWFB          TEMP

                BTFSS           _C              ; test for restore
                GOTO            Drest#v( i )

                BSF             ACC+B3,LSB
                GOTO            Dok#v( i )

Drest#v( i )    MOVFP           BARG+B1,WREG            ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B3,LSB

Dok#v( i )

                i += 1

                endw

                endm


DIV16A          macro

;       Max Timing:     15+15*18 = 285 clks
;       Min Timing:     12+15*15 = 237 clks

;       PM: 16+15*19 = 301                      DM: 7

                BTG             TEMP,LSB        ; save carry

                MOVFP           BARG+B1,WREG    ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB        ; test for restore
                GOTO            DArest0

                BSF             ACC+B2,MSB
                GOTO            DAok0
```

5

© 1994 Microchip Technology Inc.

```
DArest0         MOVFP           BARG+B1,WREG    ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B2,MSB

DAok0


                variable i = 1

                while i < 16

                BCF             _C
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP            ; save carry

                MOVFP           BARG+B1,WREG    ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB        ; test for restore
                GOTO            DArest#v( i )

                if i < 8

                BSF             ACC+B2,MSB-i    ; set bit in quotient

                else

                BSF             ACC+B3,15-i     ; set bit in quotient

                endif

                GOTO            DAok#v( i )

DArest#v( i )   MOVFP           BARG+B1,WREG    ; restore if necessary
                ADDWF           ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0


                if i < 8

                BCF             ACC+B2,MSB-i    ; clear bit in quotient

                else

                BCF             ACC+B3,15-i     ; clear bit in quotient

                endif

DAok#v( i )

                i += 1

                endw

                endm

FPD24           CLRF            TEMP,W          ; test for divide by zero
                CPFSGT          BEXP
                GOTO            SETFDZ24
```


© 1994 Microchip Technology Inc.

```
                CPFSGT          AEXP
                GOTO            RES024

D24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN24        INCF            TEMP                ; set align increment
                MOVFP           AARG+B0,ACC+B2      ; test for alignment
                MOVFP           AARG+B1,ACC+B3
                MOVFP           BARG+B1,WREG
                SUBWF           ACC+B3
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B2
                BTFSC           _C
                GOTO            DALIGN24OK

                RLCF            ACC+B1              ; align if necessary
                RLCF            ACC+B0
                DECF            TEMP                ; fix align increment

DALIGN24OK      MOVFP           BEXP,WREG           ; compute AEXP - BEXP
                SUBWF           EXP
                MOVLW           EXPBIAS
                BTFSS           _C
                GOTO            ALTB24

AGEB24          ADDWF           TEMP,W              ; if AEXP > BEXP, test for
                ADDWF           EXP                 ; overflow
                BTFSC           _C
                GOTO            SETFOV24
                GOTO            DARGOK24

ALTB24          ADDWF           TEMP,W              ; if AEXP < BEXP, test for
                ADDWF           EXP                 ; underflow
                BTFSS           _C
                GOTO            SETFUN24

DARGOK24        DIV16A                              ; macro for division

DROUND24        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B3,LSB
                GOTO            DIV24OK
                BCF             _C                  ; compute next significant bit
                RLCF            ACC+B1              ; for rounding
                RLCF            ACC+B0
                RLCF            TEMP                ; save carry

                MOVFP           BARG+B1,WREG        ; subtract
                SUBWF           ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                RRCF            TEMP                ; rotate next significant bit into
                CLRF            WREG                ; carry and add for rounding
                ADDWFC          ACC+B3
                ADDWFC          ACC+B2

                BTFSS           _C                  ; test if rounding caused carryout
                GOTO            DIV24OK
                RRCF            ACC+B2
                RRCF            ACC+B3
                INFSNZ          EXP                 ; test for overflow
                GOTO            SETFOV24
```

5

© 1994 Microchip Technology Inc.

```
DIV24OK         BTFSS           SIGN,MSB
                BCF             ACC+B2,MSB          ; clear explicit MSB if positive

                MOVFP           ACC+B2,AARG+B0      ; move result to AARG
                MOVFP           ACC+B3,AARG+B1

                RETLW           0

SETFUN24        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVPF           WREG,AEXP           ; point number = 0x 01 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG

SETFDZ24        BSF             FPFLAGS,FDZ         ; set floating point divide by zero
                RETLW           0xFF                ; flag and return error code in
        ; WREG
;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Subtract

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPS24

;       Output: 24 bit floating point difference in AEXP, AARG+B0, AARG+B1

;       Result: AARG <-  AARG - BARG

;       Max Timing:     1+116 = 117 clks                RND = 0
;                                   1+131 = 132 clks            RND = 1

;       Min Timing:     1+31 = 32 clks

;       PM: 1+212 = 213         DM: 10

FPS24           BTG             BARG+B0,MSB         ; toggle sign bit for subtraction
;-----------------------------

;       Floating Point Add

;       Input:  24 bit floating point number in AEXP, AARG+B0, AARG+B1
;               24 bit floating point number in BEXP, BARG+B0, BARG+B1

;       Use:    CALL FPA24

;       Output: 24 bit floating point sum in AEXP, AARG+B0, AARG+B1

;       Result: AARG <-  AARG - BARG

;       Max Timing:     60+17+39 = 116 clks             RND = 0
;                                   60+25+46 = 131 clks         RND = 1

;       Min Timing:     27+4 = 31 clks

;       PM: 212                                 DM: 10

FPA24           MOVFP           AARG+B0,WREG        ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVPF           WREG,TEMP
```


© 1994 Microchip Technology Inc.

```
                MOVFP   AEXP,WREG       ; use AARG if AEXP >= BEXP
                CPFSGT  BEXP
                GOTO    USEA24

USEB24          MOVFP   BARG+B0,SIGN    ; save sign in SIGN
                BSF     BARG+B0,MSB     ; make MSB's explicit
                BSF     AARG+B0,MSB

                MOVFP   AEXP,WREG       ; compute shift count in BEXP
                MOVFP   BEXP,AEXP
                SUBWF   BEXP
                BTFSC   _Z
                GOTO    BLIGNED24

                MOVLW   7
                CPFSGT  BEXP            ; do byte shift if BEXP >= 8
                GOTO    BNIB24

                SUBWF   BEXP            ; BEXP = BEXP - 7
                RLCF    AARG+B1         ; rotate next bit for rounding
                MOVFP   AARG+B0,AARG+B1
                CLRF    AARG+B0
                DCFSNZ  BEXP            ; BEXP = BEXP - 1
                GOTO    BROUND24

                CPFSGT  BEXP            ; if BEXP still >= 8, then
                GOTO    BNIB24A         ; AARG = 0 relative to BARG

                MOVFP   SIGN,AARG+B0    ; return BARG
                MOVFP   BARG+B1,AARG+B1
                RETLW   0x00

BNIB24A         MOVLW   3               ; do nibbleshift if BEXP >= 4
                CPFSGT  BEXP
                GOTO    BLOOP24A
                SUBWF   BEXP            ; BEXP = BEXP -3
                SWAPF   AARG+B1,W
                MOVPF   WREG,ACC+B4     ; save for rounding
                ANDLW   0x0F
                MOVPF   WREG,AARG+B1
                RLCF    ACC+B4          ; rotate next bit for rounding
                DCFSNZ  BEXP            ; BEXP = BEXP - 1
                GOTO    BROUND24        ; aligned if BEXP = 0

BLOOP24A        BCF     _C              ; right shift by BEXP
                RRCF    AARG+B1
                DCFSNZ  BEXP
                GOTO    BROUND24        ; aligned if BEXP = 0
                BCF     _C
                RRCF    AARG+B1
                DCFSNZ  BEXP
                GOTO    BROUND24        ; aligned if BEXP = 0
                BCF     _C              ; at most 3 right shifts are
                RRCF    AARG+B1         ; possible
                GOTO    BROUND24

BNIB24          MOVLW   3               ; do nibbleshift if BEXP >= 4
                CPFSGT  BEXP
                GOTO    BLOOP24
                SUBWF   BEXP            ; BEXP = BEXP -3
                SWAPF   AARG+B1,W
                MOVPF   WREG,ACC+B4     ; save for rounding
                ANDLW   0x0F
                MOVPF   WREG,AARG+B1
                SWAPF   AARG+B0,W
                ANDLW   0xF0
                ADDWF   AARG+B1
                SWAPF   AARG+B0,W
```

5

© 1994 Microchip Technology Inc.

```
                ANDLW           0x0F
                MOVPF           WREG,AARG+B0
                RLCF            ACC+B4              ; rotate next bit for rounding
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            BROUND24            ; aligned if BEXP = 0

BLOOP24         BCF             _C                  ; right shift by BEXP
                RRCF            AARG+B0
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24            ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B0
                RRCF            AARG+B1
                DCFSNZ          BEXP
                GOTO            BROUND24            ; aligned if BEXP = 0
                BCF             _C                  ; at most 3 right shifts are
                RRCF            AARG+B0             ; possible
                RRCF            AARG+B1

BROUND24        BTFSC           FPFLAGS,RND
                BTFSS           AARG+B1,LSB
                GOTO            BLIGNED24
                CLRF            WREG
                ADDWFC          AARG+B1             ; add carry for rounding
                ADDWFC          AARG+B0

                BTFSS           _C                  ; has rounding caused carryout?
                GOTO            BLIGNED24
                RRCF            AARG+B0             ; if so, right shift
                RRCF            AARG+B1
                INFSNZ          EXP                 ; test for floating point overflow
                GOTO            SETFOV24

BLIGNED24       BTFSS           TEMP,MSB            ; negate if signs opposite
                GOTO            AOK24
                COMF            AARG+B1
                COMF            AARG+B0
                INFSNZ          AARG+B1
                INCF            AARG+B0
                GOTO            AOK24

USEA24          MOVFP           AARG+B0,SIGN        ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make MSB's explicit
                BSF             BARG+B0,MSB

                MOVFP           BEXP,WREG           ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVPF           WREG,BEXP
                BTFSC           _Z
                GOTO            ALIGNED24

                MOVLW           7
                CPFSGT          BEXP                ; do byte shift if BEXP >= 8
                GOTO            ANIB24

                SUBWF           BEXP                ; BEXP = BEXP - 7
                RLCF            BARG+B1             ; rotate next bit for rounding
                MOVFP           BARG+B0,WREG
                MOVPF           WREG,BARG+B1
                CLRF            BARG+B0
                DCFSNZ          BEXP                ; BEXP = BEXP - 1
                GOTO            AROUND24

                CPFSGT          BEXP                ; if BEXP still >= 8, then
                GOTO            ANIB24A             ; BARG = 0 relative to AARG

                MOVFP           SIGN,AARG+B0        ; return AARG
```


© 1994 Microchip Technology Inc.

```
                RETLW   0x00

ANIB24A         MOVLW   3               ; do nibbleshift if BEXP >= 4
                CPFSGT  BEXP
                GOTO    ALOOP24A
                SUBWF   BEXP            ; BEXP = BEXP -3
                SWAPF   BARG+B1,W
                MOVPF   WREG,ACC+B4     ; save for rounding
                ANDLW   0x0F
                MOVPF   WREG,BARG+B1
                RLCF    ACC+B4          ; rotate next bit for rounding
                DCFSNZ  BEXP            ; BEXP = BEXP - 1
                GOTO    AROUND24        ; aligned if BEXP = 0

ALOOP24A        BCF     _C              ; right shift by BEXP
                RRCF    BARG+B1
                DCFSNZ  BEXP
                GOTO    AROUND24        ; aligned if BEXP = 0
                BCF     _C
                RRCF    BARG+B1
                DCFSNZ  BEXP
                GOTO    AROUND24        ; aligned if BEXP = 0
                BCF     _C              ; at most 3 right shifts are
                RRCF    BARG+B1         ; possible
                GOTO    AROUND24

ANIB24          MOVLW   3               ; do nibbleshift if BEXP >= 4
                CPFSGT  BEXP
                GOTO    ALOOP24
                SUBWF   BEXP            ; BEXP = BEXP -3
                SWAPF   BARG+B1,W
                MOVPF   WREG,ACC+B4     ; save for rounding
                ANDLW   0x0F
                MOVPF   WREG,BARG+B1
                SWAPF   BARG+B0,W
                ANDLW   0xF0
                ADDWF   BARG+B1
                SWAPF   BARG+B0,W
                ANDLW   0x0F
                MOVPF   WREG,BARG+B0
                RLCF    ACC+B4          ; rotate next bit for rounding
                DCFSNZ  BEXP            ; BEXP = BEXP - 1
                GOTO    AROUND24        ; aligned if BEXP = 0

ALOOP24         BCF     _C              ; right shift by BEXP
                RRCF    BARG+B0
                RRCF    BARG+B1
                DCFSNZ  BEXP
                GOTO    AROUND24        ; aligned if BEXP = 0
                BCF     _C
                RRCF    BARG+B0
                RRCF    BARG+B1
                DCFSNZ  BEXP
                GOTO    AROUND24        ; aligned if BEXP = 0
                BCF     _C              ; at most 3 right shifts are
                RRCF    BARG+B0         ; possible
                RRCF    BARG+B1

AROUND24        BTFSC   FPFLAGS,RND
                BTFSS   BARG+B1,LSB
                GOTO    ALIGNED24
                CLRF    WREG
                ADDWFC  BARG+B1         ; add carry for rounding
                ADDWFC  BARG+B0

                BTFSS   _C              ; has rounding caused carryout?
                GOTO    ALIGNED24
                RRCF    BARG+B0         ; if so, right shift
                RRCF    BARG+B1
```

5

```
                INFSNZ      EXP             ; test for floating point overflow
                GOTO        SETFOV24
ALIGNED24       BTFSS       TEMP,MSB        ; negate if signs opposite
                GOTO        AOK24
                COMF        BARG+B1
                COMF        BARG+B0
                INFSNZ      BARG+B1
                INCF        BARG+B0

AOK24           MOVFP       BARG+B1,WREG    ; add
                ADDWF       AARG+B1
                MOVFP       BARG+B0,WREG
                ADDWFC      AARG+B0

                BTFSC       TEMP,MSB
                GOTO        ACOMP24
                BTFSS       _C
                GOTO        FIXSIGN24

                RRCF        AARG+B0         ; shift right and increment EXP
                RRCF        AARG+B1
                INCFSZ      AEXP
                GOTO        FIXSIGN24
                GOTO        SETFOV24        ; set floating point overflow flag

ACOMP24         BTFSC       _C
                GOTO        NRM24           ; normalize and fix sign

                COMF        AARG+B1         ; negate, toggle sign bit and
                COMF        AARG+B0         ; then normalize
                INFSNZ      AARG+B1
                INCF        AARG+B0
                BTG         SIGN,MSB
                GOTO        NRM24

                END
```


© 1994 Microchip Technology Inc.

APPENDIX C

```
;       PIC16 32 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       All timings are worst case cycle counts

;         Routine               Function

;       FLO32           24 bit integer to 32 bit floating point conversion

;               Timing:             RND
;                                   0       1

;                       0       104     104
;                   SAT
;                       1       110     110

;       NRM32           normalization of unnormalized 32 bit floating point numbers

;               Timing:             RND
;                                   0       1

;                       0       90      90
;                   SAT
;                       1       96      96


;       INT32           32 bit floating point to 24 bit integer conversion

;               Timing:             RND
;                                   0       1

;                       0       90      98
;                   SAT
;                       1       90      100

;       FPA32           32 bit floating point add

;               Timing:             RND
;                                   0       1

;                       0       248     262
;                   SAT
;                       1       248     268

;       FPS32           32 bit floating point subtract

;               Timing:             RND
;                                   0       1

;                       0       250     264
;                   SAT
;                       1       250     270

;       FPM32           32 bit floating point multiply

;               Timing:             RND
;                                   0       1

;                       0       574     588
```

© 1994 Microchip Technology Inc.

```
;                   SAT
;                         1       574     591

;       FPD32             32 bit floating point divide

;               Timing:                 RND
;                                       0       1

;                         0       929     965
;                   SAT
;                         1       929     968

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       32 bit floating point representation

;       EXPONENT        8 bit biased exponent
;                       It is important to note that the use of biased exponents produces
;                       a unique representation of a floating point 0, given by
;                       EXP = HIGHBYTE = MIDBYTE = LOWBYTE = 0x00, with 0 being
;                       the only number with EXP = 0.

;       HIGHBYTE        8 bit most significant byte of sign-magnitude representation, with
;                       SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                       left of MSB

;       MIDBYTE 8 bit middle significant byte of sign-magnitude matissa

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                       RADIX
;       EXPONENT        POINT   HIGHBYTE .       MIDBYTE LOWBYTE

;       xxxxxxxx          .     Sxxxxxxx        xxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS         equ     128

;       Define library register variables

EXP             equ     0x0C    ; 8 bit biased exponent

ACC             equ     0x0D    ; most significant byte of contiguous 6 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AEXP            equ     0x0C    ; 8 bit biased exponent for argument A

AARG            equ     0x0D    ; most significant byte of mantissa for argument A

BEXP            equ     0x15    ; 8 bit biased exponent for argument B

BARG            equ     0x16    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1, AARG+B2 )  and
;               (EXP, ACC+B0, ACC+B1, ACC+B2 ) reference the same storage locations.
```


© 1994 Microchip Technology Inc.

```
;       Define floating point library exception flags

FPFLAGS         equ     0x14    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;*****************************************************************************************
;*****************************************************************************************

;       Integer to float conversion

;       Input:  24 bit 2's complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2

;       Use:    CALL    FLO32

;       Output: 32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  FLOAT( AARG )

;       Max Timing:     14+90 = 104 clks                SAT = 0
;                               14+96 = 110 clks        SAT = 1

;       Min Timing:     6+28 = 34 clks          AARG = 0
;                               6+18 = 24 clks

;       PM: 14+38 = 52          DM: 7

FLO32           MOVLW           24+EXPBIAS          ; initialize exponent and add bias
                MOVWF           EXP
                CLRF            SIGN
                BTFSS           ACC+B0,MSB          ; test sign
                GOTO            NRM32
                COMF            ACC+B2              ; if < 0, negate and set MSB in SIGN
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                BSF             SIGN,MSB

;-----------------------------

;       Normalization routine

;       Input:  32 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2, with sign in SIGN,MSB

;       Use:    CALL    NRM32

;       Output: 32 bit normalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2
```


© 1994 Microchip Technology Inc.

```
;       Result: AARG  <-  NORMALIZE( AARG )

;       Max Timing:     21+6+7*8+7 = 90 clks            SAT = 0
;                                       21+6+7*8+1+12 = 96 clks SAT = 1

;       Min Timing: 22+6 = 28 clks              AARG = 0
;                                       5+9+4 = 18 clks

;       PM: 38                                  DM: 7

NRM32           CLRF            TEMP                    ; clear exponent decrement
                MOVF            ACC+B0,W                ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM32
                MOVF            ACC+B1,W                ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                MOVF            ACC+B2,W
                MOVWF           ACC+B1
                CLRF            ACC+B2
                BSF             TEMP,3                  ; increase decrement by 8

                MOVF            ACC+B0,W                ; test if highbyte=0
                BTFSS           _Z
                GOTO            NORM32
                MOVF            ACC+B1,W                ; if so, shift 8 bits by move
                MOVWF           ACC+B0
                CLRF            ACC+B1
                BCF             TEMP,3                  ; increase decrement by 8
                BSF             TEMP,4

                MOVF            ACC+B0,W                ; if highbyte=0, result=0
                BTFSC           _Z
                GOTO            RES032

NORM32          MOVF            TEMP,W
                SUBWF           EXP
                BTFSS           _Z
                BTFSS           _C
                GOTO            SETFUN32

                BCF             _C                      ; clear carry bit

NORM32A         BTFSC           ACC+B0,MSB              ; if MSB=1, normalization done
                GOTO            FIXSIGN32
                RLF             ACC+B2                  ; otherwise, shift left and
                RLF             ACC+B1                  ; decrement EXP
                RLF             ACC+B0
                DECFSZ          EXP
                GOTO            NORM32A

                GOTO            SETFUN32                ; underflow if EXP=0

FIXSIGN32       BTFSS           SIGN,MSB
                BCF             ACC+B0,MSB              ; clear explicit MSB if positive
                RETLW           0

;*****************************************************************************************
;*****************************************************************************************

;       Float to integer conversion

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Use:    CALL    INT32

;       Output: 24 bit 2's complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2
```


© 1994 Microchip Technology Inc.

```
;       Result: AARG  <-  INT( AARG )

;       Max Timing:     26+6*7+6+16 = 90 clks          RND = 0
;                               26+6*7+6+24 = 98 clks          RND = 1, SAT = 0
;                               26+6*7+6+26 = 100 clks         RND = 1, SAT = 1

;       Min Timing:     9+6 = 15 clks          AARG = 0
;                               19+6+7 = 32 clks

;       PM: 83                                 DM: 6

INT32           MOVF            ACC+B0,W            ; save sign in SIGN
                MOVWF           SIGN
                BSF             ACC+B0,MSB          ; make MSB explicit

                MOVLW           EXPBIAS             ; remove bias from EXP
                SUBWF           EXP
                BTFSS           EXP,MSB             ; if <= 0, result=0
                BTFSC           _Z
                GOTO            RES032
                MOVF            EXP,W
                SUBLW           24
                MOVWF           EXP
                BTFSS           _Z
                BTFSC           EXP,MSB
                GOTO            SETIOV32

                MOVLW           8                   ; do byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT32
                MOVWF           EXP
                RLF             ACC+B2              ; rotate next bit for rounding
                MOVF            ACC+B1,W
                MOVWF           ACC+B2
                MOVF            ACC+B0,W
                MOVWF           ACC+B1
                CLRF            ACC+B0

                MOVLW           8                   ; do another byte shift if EXP >= 8
                SUBWF           EXP,W
                BTFSS           _C
                GOTO            SHIFT32
                MOVWF           EXP
                RLF             ACC+B2              ; rotate next bit for rounding
                MOVF            ACC+B1,W
                MOVWF           ACC+B2
                CLRF            ACC+B1

                MOVF            EXP,W               ; shift completed if EXP = 0
                BTFSC           _Z
                GOTO            SHIFT32OK

SHIFT32         BCF             _C
                RRF             ACC+B0              ; right shift by EXP
                RRF             ACC+B1
                RRF             ACC+B2
                DECFSZ          EXP
                GOTO            SHIFT32

SHIFT32OK       BTFSC           FPFLAGS,RND
                BTFSS           ACC+B2,LSB
                GOTO            INT32OK
                BTFSS           _C
                GOTO            INT32OK
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
```

5

```
                BTFSC           _Z
                INCF            ACC+B0
                BTFSC           ACC+B0,MSB          ; test for overflow
                GOTO            SETIOV32
INT32OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B0
                COMF            ACC+B1
                COMF            ACC+B2
                INCF            ACC+B2
                BTFSC           _Z
                INCF            ACC+B1
                BTFSC           _Z
                INCF            ACC+B0
                RETLW           0

RES032          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            ACC+B2
                CLRF            EXP                 ; clear EXP for other routines
                RETLW           0

SETIOV32        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two's
                BTFSS           SIGN,MSB            ; complement 16 bit integer
                MOVLW           0xFF
                MOVWF           ACC+B0              ; SIGN = 0, 0x 7F FF FF
                MOVWF           ACC+B1              ; SIGN = 1, 0x 80 00 00
                MOVWF           ACC+B2
                RLF             SIGN
                RRF             ACC+B0
                RETLW           0xFF                ; return error code in WREG


;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Multiply

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPM32

;       Output: 32 bit floating point product in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG * BARG

;       Max Timing:     26+23*22+21+21 = 574 clks       RND = 0
;                                       26+23*22+21+35 = 588 clks       RND = 1, SAT = 0
;                                       26+23*22+21+38 = 591 clks       RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks           AARG * BARG = 0
;                                       24+23*11+21+17 = 315 clks

;       PM: 94                                  DM: 14

FPM32           MOVF            AEXP,W              ; test for zero arguments
                BTFSS           _Z
                MOVF            BEXP,W
                BTFSC           _Z
                GOTO            RES032

M32BNE0         MOVF            AARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                XORWF   BARG+B0,W
                MOVWF   SIGN                    ; save sign in SIGN
                MOVF    BEXP,W
                ADDWF   EXP
                MOVLW   EXPBIAS
                BTFSS   _C
                GOTO    MTUN32

                ADDWF   EXP
                BTFSC   _C
                GOTO    SETFOV32                ; set multiply overflow flag
                GOTO    MOK32

MTUN32          ADDWF   EXP
                BTFSS   _C
                GOTO    SETFUN32

MOK32           BSF     AARG+B0,MSB             ; make argument MSB's explicit
                BSF     BARG+B0,MSB
                BCF     _C
                CLRF    ACC+B3                  ; clear initial partial product
                CLRF    ACC+B4
                CLRF    ACC+B5
                MOVLW   24
                MOVWF   TEMP                    ; initialize counter

MLOOP32         BTFSS   AARG+B2,LSB             ; test high byte
                GOTO    MNOADD32

MADD32          MOVF    BARG+B2,W
                ADDWF   ACC+B5
                MOVF    BARG+B1,W
                BTFSC   _C
                INCFSZ  BARG+B1,W
                ADDWF   ACC+B4

                MOVF    BARG+B0,W
                BTFSC   _C
                INCFSZ  BARG+B0,W
                ADDWF   ACC+B3

MNOADD32        RRF     ACC+B3
                RRF     ACC+B4
                RRF     ACC+B5
                RRF     ACC+B0
                RRF     ACC+B1
                RRF     ACC+B2
                BCF     _C
                DECFSZ  TEMP
                GOTO    MLOOP32

                BTFSC   ACC+B3,MSB              ; check for postnormalization
                GOTO    MROUND32
                RLF     ACC+B0
                RLF     ACC+B5
                RLF     ACC+B4
                RLF     ACC+B3
                DECF    EXP

MROUND32        BTFSC   FPFLAGS,RND
                BTFSS   ACC+B5,LSB
                GOTO    MUL32OK
                RLF     ACC+B0                  ; rotate next significant bit into
                BTFSC   _C
                INCF    ACC+B5                  ; carry for rounding
                BTFSC   _Z
                INCF    ACC+B4
                BTFSC   _Z
```

5

```
                INCF            ACC+B3

                BTFSS           _C                      ; has rounding caused carryout?
                GOTO            MUL32OK
                RRF             ACC+B3                  ; if so, right shift
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _C                      ; check for overflow
                GOTO            SETFOV32

MUL32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB              ; clear explicit MSB if positive

                MOVF            ACC+B3,W
                MOVWF           AARG+B0                 ; move result to AARG
                MOVF            ACC+B4,W
                MOVWF           AARG+B1
                MOVF            ACC+B5,W
                MOVWF           AARG+B2
                RETLW           0

SETFOV32        BSF             FPFLAGS,FOV             ; set floating point underflag
                BTFSS           FPFLAGS,SAT             ; test for saturation
                RETLW           0xFF                    ; return error code in WREG

                MOVLW           0xFF
                MOVWF           AEXP                    ; saturate to largest floating
                MOVWF           AARG+B0                 ; point number = 0x FF 7F FF FF
                MOVWF           AARG+B1                 ; modulo the appropriate sign bit
                MOVWF           AARG+B2
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                    ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Divide

;       Input:  32 bit floating point dividend in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point divisor in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPD32

;       Output: 32 bit floating point quotient in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG / BARG

;       Max Timing:     40+12+23*36+35+14 = 929 clks    RND = 0
;                               40+12+23*36+35+50 = 965 clks    RND = 1, SAT = 0
;                               40+12+23*36+35+53 = 968 clks    RND = 1, SAT = 1

;       Min Timing:     31+12+23*27+26+14 = 704 clks

;       PM: 152                                 DM: 14

FPD32           MOVF            BEXP,W                  ; test for divide by zero
                BTFSC           _Z
                GOTO            SETFDZ32

D32BNE0         MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN                    ; save sign in SIGN
                BSF             AARG+B0,MSB             ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN32        CLRF            TEMP                    ; clear align increment
                MOVF            AARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                MOVWF           ACC+B3              ; test for alignment
                MOVF            AARG+B1,W
                MOVWF           ACC+B4
                MOVF            AARG+B2,W
                MOVWF           ACC+B5

                MOVF            BARG+B2,W
                SUBWF           ACC+B5
                MOVF            BARG+B1,W
                BTFSS           _C
                INCFSZ          BARG+B1,W

TS1ALIGN32      SUBWF           ACC+B4
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W

TS2ALIGN32      SUBWF           ACC+B3

                CLRF            ACC+B3
                CLRF            ACC+B4
                CLRF            ACC+B5

                BTFSS           _C
                GOTO            DALIGN32OK

                BCF             _C                  ; align if necessary
                RRF             ACC+B0
                RRF             ACC+B1
                RRF             ACC+B2
                RRF             ACC+B3
                MOVLW           0x01
                MOVWF           TEMP                ; save align increment

DALIGN32OK      MOVF            BEXP,W              ; compare AEXP and BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB32

AGEB32          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV32
                GOTO            DARGOK32            ; set overflow flag

ALTB32          MOVLW           EXPBIAS
                ADDWF           TEMP,W
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN32            ; set underflow flag

DARGOK32        MOVLW           24                  ; initialize counter
                MOVWF           TEMP+B1

DLOOP32         RLF             ACC+B5              ; left shift
                RLF             ACC+B4
                RLF             ACC+B3
                RLF             ACC+B2
                RLF             ACC+B1
                RLF             ACC+B0
                RLF             TEMP

                MOVF            BARG+B2,W           ; subtract
                SUBWF           ACC+B2
                MOVF            BARG+B1,W
                BTFSS           _C
                INCFSZ          BARG+B1,W
DS132           SUBWF           ACC+B1
```


© 1994 Microchip Technology Inc.

```
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
DS232           SUBWF   ACC+B0

                RLF     BARG+B0,W
                IORWF   TEMP

                BTFSS   TEMP,LSB        ; test for restore
                GOTO    DREST32

                BSF     ACC+B5,LSB
                GOTO    DOK32

DREST32         MOVF    BARG+B2,W       ; restore if necessary
                ADDWF   ACC+B2
                MOVF    BARG+B1,W
                BTFSC   _C
                INCFSZ  BARG+B1,W
DAREST32        ADDWF   ACC+B1

                MOVF    BARG+B0,W
                BTFSC   _C
                INCF    BARG+B0,W
                ADDWF   ACC+B0

                BCF     ACC+B5,LSB

DOK32           DECFSZ  TEMP+B1
                GOTO    DLOOP32

DROUND32        BTFSC   FPFLAGS,RND
                BTFSS   ACC+B5,LSB
                GOTO    DIV32OK
                BCF     _C
                RLF     ACC+B2          ; compute next significant bit
                RLF     ACC+B1          ; for rounding
                RLF     ACC+B0
                RLF     TEMP

                MOVF    BARG+B2,W       ; subtract
                SUBWF   ACC+B2
                MOVF    BARG+B1,W
                BTFSS   _C
                INCFSZ  BARG+B1,W
                GOTO    DS1ROUND32
                BSF     _C
                BTFSS   _C
DS1ROUND32      SUBWF   ACC+B1

                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                GOTO    DS2ROUND32
                BSF     _C
                BTFSS   _C
DS2ROUND32      SUBWF   ACC+B0

                RLF     BARG+B0,W
                IORWF   TEMP,W
                ANDLW   0x01

                ADDWF   ACC+B5
                BTFSC   _C
                INCF    ACC+B4
                BTFSC   _Z
                INCF    ACC+B3
```


© 1994 Microchip Technology Inc.

```
                BTFSS           _Z                      ; test if rounding caused carryout
                GOTO            DIV32OK
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _Z                      ; test for overflow
                GOTO            SETFOV32

DIV32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB              ; clear explicit MSB if positive

                MOVF            ACC+B3,W
                MOVWF           AARG+B0                 ; move result to AARG
                MOVF            ACC+B4,W
                MOVWF           AARG+B1
                MOVF            ACC+B5,W
                MOVWF           AARG+B2

                RETLW           0

SETFUN32        BSF             FPFLAGS,FUN             ; set floating point underflag
                BTFSS           FPFLAGS,SAT             ; test for saturation
                RETLW           0xFF                    ; return error code in WREG

                MOVLW           0x01                    ; saturate to smallest floating
                MOVWF           AEXP                    ; point number = 0x 01 00 00 00
                CLRF            AARG+B0                 ; modulo the appropriate sign bit
                CLRF            AARG+B1
                CLRF            AARG+B2
                RLF             SIGN
                RRF             AARG+B0
                RETLW           0xFF                    ; return error code in WREG

SETFDZ32        BSF             FPFLAGS,FDZ             ; set divide by zero flag
                RETLW           0xFF

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Subtract

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPS32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     2+248 = 250 clks                RND = 0
;                       2+262 = 264 clks                RND = 1, SAT = 0
;                       2+268 = 270 clks                RND = 1, SAT = 1

;       Min Timing:     2+50 = 52 clks

;       PM: 2+136 = 138         DM: 14

FPS32           MOVLW           0x80
                XORWF           BARG+B0

;-------------------------------

;       Floating Point Add

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2
```

5

```
;       Use:    CALL FPA32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     31+38+6*7+6+41+90 = 248 clks    RND = 0
;                               31+38+6*7+6+55+90 = 262 clks    RND = 1, SAT = 0
;                               31+38+6*7+6+55+96 = 268 clks    RND = 1, SAT = 1

;       Min Timing:     8+38+4 = 50 clks

;       PM: 136                         DM: 14

FPA32           MOVF        AARG+B0,W       ; exclusive or of signs in TEMP
                XORWF       BARG+B0,W
                MOVWF       TEMP

                MOVF        AEXP,W          ; use AARG if AEXP >= BEXP
                SUBWF       BEXP,W
                BTFSS       _C
                GOTO        USEA32

                MOVF        BEXP,W
                MOVWF       ACC+B5          ; otherwise, swap AARG and BARG
                MOVF        AEXP,W
                MOVWF       BEXP
                MOVF        ACC+B5,W
                MOVWF       AEXP

                MOVF        BARG+B0,W
                MOVWF       ACC+B5
                MOVF        AARG+B0,W
                MOVWF       BARG+B0
                MOVF        ACC+B5,W
                MOVWF       AARG+B0

                MOVF        BARG+B1,W
                MOVWF       ACC+B5
                MOVF        AARG+B1,W
                MOVWF       BARG+B1
                MOVF        ACC+B5,W
                MOVWF       AARG+B1

                MOVF        BARG+B2,W
                MOVWF       ACC+B5
                MOVF        AARG+B2,W
                MOVWF       BARG+B2
                MOVF        ACC+B5,W
                MOVWF       AARG+B2

USEA32          MOVF        AARG+B0,W
                MOVWF       SIGN            ; save sign in SIGN
                BSF         AARG+B0,MSB     ; make MSB's explicit
                BSF         BARG+B0,MSB

                MOVF        BARG+B0,W
                MOVWF       ACC+B3
                MOVF        BARG+B1,W
                MOVWF       ACC+B4
                MOVF        BARG+B2,W
                MOVWF       ACC+B5

                MOVF        BEXP,W          ; compute shift count in BEXP
                SUBWF       AEXP,W
                MOVWF       BEXP
                BTFSC       _Z
                GOTO        AROUND32
```


© 1994 Microchip Technology Inc.

```
                MOVLW           8
                SUBWF           BEXP,W
                BTFSS           _C                      ; if BEXP >= 8, do byte shift
                GOTO            ALIGNB32
                MOVWF           BEXP
                RLF             ACC+B5                  ; rotate next bit for rounding
                MOVF            ACC+B4,W
                MOVWF           ACC+B5
                MOVF            ACC+B3,W
                MOVWF           ACC+B4
                CLRF            ACC+B3

                MOVLW           8
                SUBWF           BEXP,W
                BTFSS           _C                      ; if BEXP >= 8, do byte shift
                GOTO            ALIGNB32
                MOVWF           BEXP
                RLF             ACC+B5                  ; rotate next bit for rounding
                MOVF            ACC+B4,W
                MOVWF           ACC+B5
                CLRF            ACC+B4


ALIGNB32        MOVF            BEXP,W                  ; already aligned if BEXP = 0
                BTFSC           _Z
                GOTO            AROUND32

ALOOPB32        BCF             _C                      ; right shift by BEXP
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                DECFSZ          BEXP
                GOTO            ALOOPB32

AROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            ALIGNED32

                BTFSS           _C
                GOTO            ALIGNED32
                INCF            ACC+B5
                BTFSC           _Z
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3

                BTFSS           _Z
                GOTO            ALIGNED32
                RRF             ACC+B3
                RRF             ACC+B4
                RRF             ACC+B5
                INCF            EXP
                BTFSC           _Z
                GOTO            SETFOV32

ALIGNED32       BTFSS           TEMP,MSB                ; negate if signs opposite
                GOTO            AOK32

                COMF            ACC+B3
                COMF            ACC+B4
                COMF            ACC+B5
                INCF            ACC+B5
                BTFSC           _Z
                INCF            ACC+B4
                BTFSC           _Z
                INCF            ACC+B3

AOK32           MOVF            ACC+B5,W                ; add
```


© 1994 Microchip Technology Inc.

```
                ADDWF           AARG+B2
                MOVF            ACC+B4,W
                BTFSC           _C
                INCFSZ          ACC+B4,W
                ADDWF           AARG+B1

                MOVF            ACC+B3,W
                BTFSC           _C
                INCFSZ          ACC+B3,W
                ADDWF           AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP32
                BTFSS           _C
                GOTO            FIXSIGN32

                RRF             AARG+B0         ; shift right and increment EXP
                RRF             AARG+B1
                RRF             AARG+B2
                INCFSZ           AEXP
                GOTO            FIXSIGN32
                GOTO            SETFOV32

ACOMP32         BTFSC           _C
                GOTO            NRM32           ; normalize and fix sign

                COMF            AARG+B0         ; negate, toggle sign bit and
                COMF            AARG+B1         ; then normalize
                COMF            AARG+B2
                INCF            AARG+B2
                BTFSC           _Z
                INCF            AARG+B1
                BTFSC           _Z
                INCF            AARG+B0

                MOVLW           0x80
                XORWF           SIGN
                GOTO            NRM32

                END
```


© 1994 Microchip Technology Inc.

APPENDIX D

```
;       PIC17 32 BIT FLOATING POINT LIBRARY     VERSION 1.14

;       Unary operations: both input and output are in AEXP,AARG

;       Binary operations: input in AEXP,AARG and BEXP,BARG with output in AEXP,AARG

;       All routines return WREG = 0x00 for successful completion, and WREG = 0xFF
;       for an error condition specified in FPFLAGS.

;       Max timings are worst case cycle counts, while Min timings are non-exception
;       best case cycle counts.

;         Routine               Function

;       FLO24           16 bit integer to 24 bit floating point conversion

;               Timing:             RND
;                                   0       1

;                           0       60      60
;                   SAT
;                           1       67      67

;       NRM24           normalization of unnormalized 24 bit floating point numbers

;               Timing:             RND
;                                   0       1

;                           0       48      48
;                   SAT
;                           1       55      55


;       INT24           24 bit floating point to 16 bit integer conversion

;               Timing:             RND
;                                   0       1

;                           0       69      73
;                   SAT
;                           1       69      76

;       FPA24           24 bit floating point add

;               Timing:             RND
;                                   0       1

;                           0       144     159
;                   SAT
;                           1       144     159

;       FPS24           24 bit floating point subtract

;               Timing:             RND
;                                   0       1

;                           0       145     160
;                   SAT
;                           1       145     160

;       FPM24           24 bit floating point multiply

;               Timing:             RND
;                                   0       1

;                           0       320     331
```

© 1994 Microchip Technology Inc.

```
;                   SAT
;                         1       320     336

;       FPD24             24 bit floating point divide

;                 Timing:             RND
;                                     0       1

;                         0       597     621
;                   SAT
;                         1       597     627


                  list    r=dec,x=on,t=off,p=17C42

                  include <PIC17.INC>

;*****************************************************************************************
;*****************************************************************************************

;       32 bit floating point representation

;       EXPONENT          8 bit biased exponent
;                         It is important to note that the use of biased exponents produces
;                         a unique representation of a floating point 0, given by
;                         EXP = HIGHBYTE = MIDBYTE = LOWBYTE = 0x00, with 0 being
;                         the only number with EXP = 0.

;       HIGHBYTE          8 bit most significant byte of sign-magnitude representation, with
;                         SIGN = MSB, and implicit mantissa MSB = 1 and radix point to the
;                         left of MSB

;       MIDBYTE 8 bit middle significant byte of sign-magnitude matissa

;       LOWBYTE 8 bit least significant byte of sign-magnitude matissa

;                         RADIX
;       EXPONENT          POINT   HIGHBYTE        MIDBYTE LOWBYTE

;       xxxxxxxx            .     Sxxxxxxx        xxxxxxxx        xxxxxxxx


;       Define literal constants

EXPBIAS           equ     128

;       Define library register variables

EXP               equ     0x18    ; 8 bit biased exponent

ACC               equ     0x19    ; most significant byte of contiguous 6 byte accumulator

SIGN              equ     0x1F    ; save location for sign in MSB

TEMP              equ     0x25    ; temporary storage

;       Define binary operation arguments

AEXP              equ     0x18    ; 8 bit biased exponent for argument A

AARG              equ     0x19    ; most significant byte of mantissa for argument A

BEXP              equ     0x21    ; 8 bit biased exponent for argument B

BARG              equ     0x22    ; most significant byte of mantissa for argument B

;       Note:   (AEXP, AARG+B0, AARG+B1 , AARG+B2)  and
;               (EXP, ACC+B0, ACC+B1 , ACC+B2) reference the same storage locations.
```


© 1994 Microchip Technology Inc.

```
;       Define floating point library exception flags

FPFLAGS         equ     0x20    ; floating point library exception flags

IOV             equ     0       ; bit0 = integer overflow flag

FOV             equ     1       ; bit1 = floating point overflow flag

FUN             equ     2       ; bit2 = floating point underflow flag

FDZ             equ     3       ; bit3 = floating point divide by zero flag

RND             equ     6       ; bit6 = floating point rounding flag, 0 = truncation
                                ; 1 = rounding to nearest LSB

SAT             equ     7       ; bit7 = floating point saturate flag, 0 = terminate on
                                ; exception without saturation, 1 = terminate on
                                ; exception with saturation to appropriate value

;*****************************************************************************************
;*****************************************************************************************

;       Integer to float conversion

;       Input:  24 bit 2's complement integer right justified in AARG+B0,AARG+B1,
;                               AARG+B2

;       Use:    CALL    FLO32

;       Output: 32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  FLOAT( AARG )

;       Max Timing:     12+48 = 60 clks         SAT = 0
;                                       12+55 = 67 clks         SAT = 1

;       Min Timing:     6+24 = 30 clks          AARG = 0
;                                       6+20 = 26 clks

;       PM: 12+108 = 120                DM: 7

FLO32           MOVLW           24+EXPBIAS          ; initialize exponent and add bias
                MOVWF           EXP
                MOVFP           ACC+B0,SIGN         ; save sign in SIGN
                BTFSS           ACC+B0,MSB          ; test sign
                GOTO            NRM32
                CLRF            WREG                ; if < 0, negate, set MSB in SIGN
                COMF            ACC+B2
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0

;_____________________________________

;       Normalization routine

;       Input:  32 bit unnormalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2 with sign in SIGN,MSB.

;       Use:    CALL    NRM32

;       Output: 32 bit normalized floating point number in AEXP, AARG+B0, AARG+B1,
;               AARG+B2

;       Result: AARG  <-  NORMALIZE( AARG )

;       Max Timing:     21+19+8 = 48 clks               SAT = 0
```


© 1994 Microchip Technology Inc.

```
;                                       21+19+15 = 55 clks              SAT = 1

;       Min Timing:     4+7+7+6 = 24 clks               AARG = 0
;                               3+5+4+8 = 20 clks

;       PM: 108                                 DM: 7

NRM32           CLRF            TEMP,W          ; clear exponent decrement
                CPFSGT          ACC+B0          ; test if highbyte=0
                GOTO            NRM32A

TNIB32          MOVLW           0xF0            ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32
                SWAPF           ACC+B0          ; if so, shift 4 bits
                SWAPF           ACC+B1,W
                ANDLW           0x0F
                ADDWF           ACC+B0

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B1
                SWAPF           ACC+B2,W
                ANDLW           0x0F
                ADDWF           ACC+B1

                SWAPF           ACC+B2,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B2

                BSF             TEMP,2          ; increase decrement by 4

NORM32          BCF             _C              ; clear carry bit

                BTFSC           ACC+B0,MSB      ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B2          ; otherwise, shift left and
                RLCF            ACC+B1          ; increment decrement
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB      ; since highnibble != 0, at most
                GOTO            TNORMUN32       ; 3 left shifts are required
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP

TNORMUN32       MOVFP           TEMP,WREG       ; if EXP <= decrement in TEMP,
                CPFSGT          EXP             ; floating point underflow has
                GOTO            SETFUN32        ; occured
                SUBWF           EXP             ; otherwise, compute EXP

FIXSIGN32       BTFSS           SIGN,MSB
                BCF             ACC+B0,MSB      ; clear explicit MSB if positive
                RETLW           0

NRM32A          MOVFP           ACC+B1,ACC+B0   ; shift 8 bits by move
                MOVFP           ACC+B2,ACC+B1
                CLRF            ACC+B2,W
                BSF             TEMP,3          ; increase decrement by 8
                CPFSGT          ACC+B0          ; test if highbyte=0
                GOTO            NRM32B
```


© 1994 Microchip Technology Inc.

```
TNIB32A         MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32A
                SWAPF           ACC+B0              ; if so, shift 4 bits
                SWAPF           ACC+B1,W
                ANDLW           0x0F
                ADDWF           ACC+B0

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                MOVPF           WREG,ACC+B1
                BSF             TEMP,2              ; increase decrement by 4

NORM32A         BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B1              ; otherwise, shift left and
                RLCF            ACC+B0              ; increment decrement
                INCF            TEMP
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN32           ; 3 left shifts are required
                RLCF            ACC+B1
                RLCF            ACC+B0
                INCF            TEMP
                GOTO            TNORMUN32

NRM32B          MOVFP           ACC+B1,ACC+B0       ; shift 8 bits by move
                CLRF            ACC+B1,W
                BCF             TEMP,3              ; increase decrement by 8
                BSF             TEMP,4
                CPFSGT          ACC+B0              ; if highbyte=0, result=0
                GOTO            RES032

TNIB32B         MOVLW           0xF0                ; test if highnibble=0
                ANDWF           ACC+B0,W
                TSTFSZ          WREG
                GOTO            NORM32B
                SWAPF           ACC+B0              ; if so, shift 4 bits

                BSF             TEMP,2              ; increase decrement by 4

NORM32B         BCF             _C                  ; clear carry bit

                BTFSC           ACC+B0,MSB          ; if MSB=1, normalization done
                GOTO            TNORMUN32
                RLCF            ACC+B0              ; otherwise, shift left and
                INCF            TEMP                ; increment decrement
                BTFSC           ACC+B0,MSB
                GOTO            TNORMUN32
                RLCF            ACC+B0
                INCF            TEMP
                BTFSC           ACC+B0,MSB          ; since highnibble != 0, at most
                GOTO            TNORMUN32           ; 3 left shifts are required
                RLCF            ACC+B0
                INCF            TEMP
                GOTO            TNORMUN32

;*****************************************************************************************
;*****************************************************************************************
```

5

```
;       Float to integer conversion

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Use:    CALL    INT32

;       Output: 24 bit 2's complement integer right justified in AARG+B0, AARG+B1,
;               AARG+B2

;       Result: AARG  <-  INT( AARG )

;       Max Timing:     11+43+15 = 69 clks              RND = 0
;                                                   11+43+19 = 73 clks          RND = 1, SAT = 0
;                                                   11+43+22 = 76 clks          RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks               AARG = 0
;                                                   11+11+8 = 30 clks

;       PM: 155                                     DM: 7

INT32           MOVFP       ACC+B0,SIGN         ; save sign in SIGN
                BSF         ACC+B0,MSB          ; make MSB explicit

                MOVLW       EXPBIAS             ; remove bias from EXP
                CPFSGT      EXP                 ; if <= 0, result=0
                GOTO        RES032
                SUBWF       EXP

                MOVLW       24
                CPFSLT      EXP                 ; if >= 16, integer overflow
                GOTO        SETIOV32            ; will occur
                SUBWF       EXP,W
                NEGW        EXP

                MOVLW       7                   ; do byte shift if EXP >= 8
                CPFSGT      EXP
                GOTO        SNIB32
                SUBWF       EXP                 ; EXP = EXP - 7
                RLCF        ACC+B2              ; rotate next bit for rounding
                MOVFP       ACC+B1,ACC+B2
                MOVFP       ACC+B0,ACC+B1
                CLRF        ACC+B0
                DCFSNZ      EXP                 ; EXP = EXP - 1
                GOTO        SHIFT32OK           ; shift completed if EXP = 0

                CPFSGT      EXP                 ; do another byte shift if EXP >= 8
                GOTO        SNIB32A
                SUBWF       EXP                 ; EXP = EXP - 7
                RLCF        ACC+B2              ; rotate next bit for rounding
                MOVFP       ACC+B1,ACC+B2
                CLRF        ACC+B1
                DCFSNZ      EXP                 ; EXP = EXP - 1
                GOTO        SHIFT32OK           ; shift completed if EXP = 0

SNIB32B         MOVLW       3                   ; do nibble shift if EXP >= 4
                CPFSGT      EXP
                GOTO        SHIFT32B
                SUBWF       EXP                 ; EXP = EXP - 3
                SWAPF       ACC+B2,W
                MOVPF       WREG,ACC+B3         ; save for rounding
                ANDLW       0x0F
                MOVPF       WREG,ACC+B2
                RLCF        ACC+B3              ; rotate next bit for rounding

                DCFSNZ      EXP                 ; EXP = EXP - 1
                GOTO        SHIFT32OK           ; shift completed if EXP = 0

SHIFT32B        BCF         _C                  ; at most 3 right shifts are required
```


© 1994 Microchip Technology Inc.

```
                RRCF            ACC+B2              ; right shift by EXP
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B2
                GOTO            SHIFT32OK

SNIB32A         MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT32A
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B2,W
                MOVPF           WREG,ACC+B3         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                ADDWF           ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1
                RLCF            ACC+B3              ; rotate next bit for rounding

                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT32OK           ; shift completed if EXP = 0

SHIFT32A        BCF             _C                  ; at most 3 right shifts are required
                RRCF            ACC+B1              ; right shift by EXP
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B1
                RRCF            ACC+B2
                GOTO            SHIFT32OK

SNIB32          MOVLW           3                   ; do nibble shift if EXP >= 4
                CPFSGT          EXP
                GOTO            SHIFT32
                SUBWF           EXP                 ; EXP = EXP - 3
                SWAPF           ACC+B2,W
                MOVPF           WREG,ACC+B3         ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0xF0
                ADDWF           ACC+B2

                SWAPF           ACC+B1,W
                ANDLW           0x0F
                MOVPF           WREG,ACC+B1

                SWAPF           ACC+B0,W
                ANDLW           0xF0
                ADDWF           ACC+B1

                SWAPF           ACC+B0,W
```


© 1994 Microchip Technology Inc.

```
                ANDLW           0x0F
                MOVPF           WREG,ACC+B0
                RLCF            ACC+B3              ; rotate next bit for rounding

                DCFSNZ          EXP                 ; EXP = EXP - 1
                GOTO            SHIFT32OK           ; shift completed if EXP = 0

SHIFT32         BCF             _C                  ; at most 3 right shifts are required
                RRCF            ACC+B0              ; right shift by EXP
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                RRCF            ACC+B2
                DCFSNZ          EXP
                GOTO            SHIFT32OK           ; shift completed if EXP = 0
                BCF             _C
                RRCF            ACC+B0
                RRCF            ACC+B1
                RRCF            ACC+B2

SHIFT32OK       CLRF            WREG
                BTFSC           FPFLAGS,RND
                BTFSS           ACC+B2,LSB
                GOTO            INT32OK
                ADDWFC          ACC+B2              ; add next bit for rounding
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0
                BTFSC           ACC+B0,MSB
                GOTO            SETIOV32

INT32OK         BTFSS           SIGN,MSB            ; if sign bit set, negate
                RETLW           0
                COMF            ACC+B2
                COMF            ACC+B1
                COMF            ACC+B0
                INCF            ACC+B2
                ADDWFC          ACC+B1
                ADDWFC          ACC+B0
                RETLW           0

RES032          CLRF            ACC+B0              ; integer result equals zero
                CLRF            ACC+B1
                CLRF            ACC+B2
                CLRF            EXP                 ; clear exponent for other routines
                RETLW           0

SETIOV32        BSF             FPFLAGS,IOV         ; set integer overflow flag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                CLRF            ACC+B0              ; saturate to largest two's
                BTFSS           SIGN,MSB            ; complement 24 bit integer
                SETF            ACC+B0              ; SIGN = 0, 0x 7F FF FF
                MOVPF           ACC+B0,ACC+B1       ; SIGN = 1, 0x 80 00 00
                MOVPF           ACC+B0,ACC+B2
                RLCF            SIGN
                RRCF            ACC+B0
                RETLW           0xFF                ; return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Multiply

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
```


© 1994 Microchip Technology Inc.

```
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPM32

;       Output: 32 bit floating point product in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG * BARG

;       Max Timing:     19+283+18 = 320 clks            RND = 0, SAT = 0
;                               19+283+29 = 331 clks            RND = 1, SAT = 0
;                               19+283+34 = 336 clks            RND = 1, SAT = 1

;       Min Timing:     5+6 = 11 clks           AARG * BARG = 0
;                               18+54+16 = 88 clks

;       PM: 21+322+39 = 382                     DM: 13

MUL32           macro

;       Max Timing:     13+22*12+6 = 283 clks

;       Min Timing:     4+22*2+6 = 54 clks

;       PM: 4+24*2+6+22*12 = 322                DM: 9

                variable i = 0
                variable j = 1

                BCF             _C              ; clear carry bit

                CLRF            ACC+B3          ; clear initial partial product
                CLRF            ACC+B4
                CLRF            ACC+B5

                while   i < 24

                if      i < 8

                BTFSC           AARG+B2,i       ; test low byte

                endif

                if      i >= 8 && i < 16

                BTFSC           AARG+B1,i-8     ; test middle byte

                endif

                GOTO            Madd32#v( i )

                if      i >= 16

                BTFSC           AARG+B0,i-16    ; test high byte

                endif

                GOTO            Madd32#v( i )

                i += 1

                endw

;       we can not get here since the MSB's are explicit

Madd320         MOVFP           BARG+B0,ACC+B3  ; if we get here, ACC=0 so move
                MOVFP           BARG+B1,ACC+B4  ; BARG for add
                MOVFP           BARG+B2,ACC+B5

                RRCF            ACC+B3
```


© 1994 Microchip Technology Inc.

```
                RRCF            ACC+B4
                RRCF            ACC+B5

                while j < 23
                if      j < 8

                BTFSS           AARG+B2,j           ; test low byte

                endif
                if      j >= 8 && j < 16

                BTFSS           AARG+B1,j-8         ; test middle byte

                endif

                if      j >= 16

                BTFSS           AARG+B0,j-16        ; test middle byte

                endif

                GOTO            Mnoadd32#v( j )

Madd32#v( j )   MOVFP           BARG+B2,WREG
                ADDWF           ACC+B5              ; add low byte
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B4              ; add middle byte
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B3              ; add high byte

Mnoadd32#v( j ) RRCF            ACC+B3
                RRCF            ACC+B4
                RRCF            ACC+B5

                if j > 21

                RRCF            ACC+B1              ; save bits for rounding

                endif

                if j < 22

                BCF             _C                  ; clear carry when rotate possible

                endif

                j += 1

                endw


Madd3223        MOVFP           BARG+B2,WREG
                ADDWF           ACC+B5              ; add low byte
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B4              ; add middle byte
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B3              ; add high byte

                endm

FPM32           CLRF            WREG                ; test for zero arguments
                CPFSEQ          BEXP
                CPFSGT          AEXP
                GOTO            RES032

M24BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                MOVPF   WREG,SIGN       ; save sign in SIGN

                MOVFP   BEXP,WREG
                ADDWF   EXP
                MOVLW   EXPBIAS
                BTFSS   _C
                GOTO    MTUN32

                ADDWF   EXP             ; remove bias and overflow test
                BTFSC   _C
                GOTO    SETFOV32
                GOTO    MOK32

MTUN32          ADDWF   EXP             ; remove bias and underflow test
                BTFSS   _C
                GOTO    SETFUN32

MOK32           BSF     AARG+B0,MSB     ; make argument MSB's explicit
                BSF     BARG+B0,MSB

                MUL32                   ; use macro for multiplication

                BTFSC   _C              ; check for postnormalization
                GOTO    MRIGHT32
                DECF    EXP
                GOTO    MROUND32
MRIGHT32        RRCF    ACC+B3
                RRCF    ACC+B4
                RRCF    ACC+B5
                RRCF    ACC+B1

MROUND32        BTFSC   FPFLAGS,RND
                BTFSS   ACC+B5,LSB
                GOTO    MUL32OK
                CLRF    WREG
                RLCF    ACC+B1          ; rotate next significant bit into
                ADDWFC  ACC+B5          ; carry for rounding
                ADDWFC  ACC+B4
                ADDWFC  ACC+B3

                BTFSS   _C              ; has rounding caused carryout?
                GOTO    MUL32OK
                RRCF    ACC+B3          ; if so, right shift
                RRCF    ACC+B4
                RRCF    ACC+B5
                INFSNZ  EXP             ; test for floating point overflow
                GOTO    SETFOV32

MUL32OK         BTFSS   SIGN,MSB
                BCF     ACC+B3,MSB      ; clear explicit MSB if positive

                MOVFP   ACC+B3,AARG+B0  ; move result to AARG
                MOVFP   ACC+B4,AARG+B1
                MOVFP   ACC+B5,AARG+B2
                RETLW   0

SETFOV32        BSF     FPFLAGS,FOV     ; set floating point underflag
                BTFSS   FPFLAGS,SAT     ; test for saturation
                RETLW   0xFF            ; return error code in WREG

                SETF    AEXP            ; saturate to largest floating
                SETF    AARG+B0         ; point number = 0x FF 7F FF FF
                SETF    AARG+B1         ; modulo the appropriate sign bit
                SETF    AARG+B2
                RLCF    SIGN
                RRCF    AARG+B0
                RETLW   0xFF            ; return error code in WREG
```

5

```
;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Divide

;       Input:  32 bit floating point dividend in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point divisor in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL    FPD32

;       Output: 32 bit floating point quotient in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG / BARG

;       Max Timing:     27+11+548+11 = 597 clks RND = 0
;                       27+11+548+35 = 621 clks RND = 1, SAT = 0
;                       27+11+548+41 = 627 clks RND = 1, SAT = 1

;       Min Timing:     6+6 = 12 clks           AARG = 0
;                       33+428+11 = 472 clks

;       PM: 42+572+47 = 661                     DM: 14

DIV24           macro

;       Timing: 1+24*26 = 625 clks

                variable i = 0

                BCF             _C

                while i < 24

                CLRF            TEMP                    ; left shift
                RLCF            ACC+B5
                RLCF            ACC+B4
                RLCF            ACC+B3
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP

                MOVFP           BARG+B2,WREG            ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                CLRF            WREG
                SUBWFB          TEMP

                BTFSS           _C                      ; test for restore
                GOTO            Drest#v( i )

                BSF             ACC+B5,LSB
                GOTO            Dok#v( i )

Drest#v( i )    MOVFP           BARG+B2,WREG            ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B5,LSB

Dok#v( i )

                i += 1
```


© 1994 Microchip Technology Inc.

```
                endw

                endm

DIV24A          macro

;       Min Timing:     19+23*23 = 548 clks

;       Min Timing:     14+23*18 = 428 clks

;       PM: 20+23*24 = 572                      DM: 9

                BTG             TEMP,LSB        ; save carry

                MOVFP           BARG+B2,WREG    ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB        ; test for restore
                GOTO            DArest0

                BSF             ACC+B3,MSB
                GOTO            DAok0

DArest0         MOVFP           BARG+B2,WREG    ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0
                BCF             ACC+B3,MSB

DAok0

                variable i = 1

                while i < 24

                BCF             _C
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP            ; save carry

                MOVFP           BARG+B2,WREG    ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                BTFSS           TEMP,LSB        ; test for restore
                GOTO            DArest#v( i )

                if i < 8

                BSF             ACC+B3,MSB-i    ; set bit in quotient

                endif
```


© 1994 Microchip Technology Inc.

```
                if i >= 8 && i < 16

                BSF             ACC+B4,15-i         ; set bit in quotient

                endif

                if i >= 16

                BSF             ACC+B5,23-i         ; set bit in quotient

                endif

                GOTO            DAok#v( i )
DArest#v( i )   MOVFP           BARG+B2,WREG        ; restore if necessary
                ADDWF           ACC+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          ACC+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          ACC+B0


                if i < 8

                BCF             ACC+B3,MSB-i        ; clear bit in quotient

                endif

                if i >= 8 && i < 16

                BCF             ACC+B4,15-i         ; clear bit in quotient

                endif

                if i >= 16

                BCF             ACC+B5,23-i         ; clear bit in quotient

                endif

DAok#v( i )

                i += 1

                endw

                endm

FPD32           CLRF            TEMP,W
                CPFSGT  BEXP                        ; test for divide by zero
                GOTO            SETFDZ32

                CPFSGT  AEXP
                GOTO            RES032

D32BNE0         MOVFP           AARG+B0,WREG
                XORWF           BARG+B0,W
                MOVPF           WREG,SIGN           ; save sign in SIGN
                BSF             AARG+B0,MSB         ; make argument MSB's explicit
                BSF             BARG+B0,MSB

TALIGN32        INCF            TEMP                ; set align increment
                MOVFP           AARG+B0,ACC+B3      ; test for alignment
                MOVFP           AARG+B1,ACC+B4
                MOVFP           AARG+B2,ACC+B5
                MOVFP           BARG+B2,WREG
                SUBWF           ACC+B5
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B4
```


© 1994 Microchip Technology Inc.

```
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B3
                BTFSC           _C
                GOTO            DALIGN32OK

                BCF             _C                      ; align if necessary
                RLCF            ACC+B2
                RLCF            ACC+B1
                RLCF            ACC+B0
                DECF            TEMP                    ; fix align increment

DALIGN32OK      MOVFP           BEXP,WREG               ; compute AEXP - BEXP
                SUBWF           EXP
                BTFSS           _C
                GOTO            ALTB32

AGEB32          MOVLW           EXPBIAS                 ; if AEXP > BEXP, test for
                ADDWF           TEMP,W                  ; overflow
                ADDWF           EXP
                BTFSC           _C
                GOTO            SETFOV32
                GOTO            DARGOK32

ALTB32          MOVLW           EXPBIAS                 ; if AEXP < BEXP, test for
                ADDWF           TEMP,W                  ; underflow
                ADDWF           EXP
                BTFSS           _C
                GOTO            SETFUN32

DARGOK32        DIV24A                                  ; macro for division

DROUND32        BTFSC           FPFLAGS,RND
                BTFSS           ACC+B5,LSB
                GOTO            DIV32OK
                BCF             _C                      ; compute next significant bit
                RLCF            ACC+B2                  ; for rounding
                RLCF            ACC+B1
                RLCF            ACC+B0
                RLCF            TEMP                    ; save carry

                MOVFP           BARG+B2,WREG            ; subtract
                SUBWF           ACC+B2
                MOVFP           BARG+B1,WREG
                SUBWFB          ACC+B1
                MOVFP           BARG+B0,WREG
                SUBWFB          ACC+B0
                RLCF            WREG
                IORWF           TEMP

                RRCF            TEMP                    ; rotate next significant bit into
                CLRF            WREG                    ; carry and add for rounding
                ADDWFC          ACC+B5
                ADDWFC          ACC+B4
                ADDWFC          ACC+B3

                BTFSS           _C                      ; test if rounding caused carryout
                GOTO            DIV32OK
                RRCF            ACC+B3
                RRCF            ACC+B4
                RRCF            ACC+B5
                INFSNZ          EXP                     ; test for overflow
                GOTO            SETFOV32

DIV32OK         BTFSS           SIGN,MSB
                BCF             ACC+B3,MSB              ; clear explicit MSB if positive

                MOVFP           ACC+B3,AARG+B0          ; move result to AARG
                MOVFP           ACC+B4,AARG+B1
                MOVFP           ACC+B5,AARG+B2
```

5

```
                RETLW           0

SETFUN32        BSF             FPFLAGS,FUN         ; set floating point underflag
                BTFSS           FPFLAGS,SAT         ; test for saturation
                RETLW           0xFF                ; return error code in WREG

                MOVLW           0x01                ; saturate to smallest floating
                MOVPF           WREG,AEXP           ; point number = 0x 01 00 00 00
                CLRF            AARG+B0             ; modulo the appropriate sign bit
                CLRF            AARG+B1
                CLRF            AARG+B2
                RLCF            SIGN
                RRCF            AARG+B0
                RETLW           0xFF                ; return error code in WREG

SETFDZ32        BSF             FPFLAGS,FDZ         ; set floating point divide by zero
                RETLW           0xFF                ; flag and return error code in WREG

;*****************************************************************************************
;*****************************************************************************************

;       Floating Point Subtract

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPS32

;       Output: 32 bit floating point difference in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     1+144 = 145 clks            RND = 0
;                       1+159 = 160 clks            RND = 1

;       Min Timing:     1+34 = 35 clks

;       PM: 330                         DM: 14

FPS32           BTG             BARG+B0,MSB         ; toggle sign bit for subtraction

;-----------------------------------

;       Floating Point Add

;       Input:  32 bit floating point number in AEXP, AARG+B0, AARG+B1, AARG+B2
;               32 bit floating point number in BEXP, BARG+B0, BARG+B1, BARG+B2

;       Use:    CALL FPA32

;       Output: 32 bit floating point sum in AEXP, AARG+B0, AARG+B1, AARG+B2

;       Result: AARG  <-  AARG - BARG

;       Max Timing:     7+67+22+48 = 144 clks       RND = 0
;                       7+75+22+55 = 159 clks       RND = 1

;       Min Timing:     6+23+5 = 34 clks

;       PM: 329                         DM: 14

FPA32           MOVFP           AARG+B0,WREG        ; exclusive or of signs in TEMP
                XORWF           BARG+B0,W
                MOVPF           WREG,TEMP

                MOVFP           AEXP,WREG           ; use AARG if AEXP >= BEXP
                CPFSGT          BEXP
                GOTO            USEA32
```


© 1994 Microchip Technology Inc.

```
USEB32      MOVFP     BARG+B0,SIGN      ; save sign in SIGN
            BSF       BARG+B0,MSB       ; make MSB's explicit
            BSF       AARG+B0,MSB

            MOVFP     AEXP,WREG         ; compute shift count in BEXP
            MOVFP     BEXP,AEXP
            SUBWF     BEXP
            BTFSC     _Z
            GOTO      BLIGNED32

            MOVLW     7
            CPFSGT    BEXP              ; do byte shift if BEXP >= 8
            GOTO      BNIB32
            SUBWF     BEXP              ; BEXP = BEXP - 7
            RLCF      AARG+B2           ; rotate next bit for rounding
            MOVFP     AARG+B1,AARG+B2
            MOVFP     AARG+B0,AARG+B1
            CLRF      AARG+B0
            DCFSNZ    BEXP              ; BEXP = BEXP - 1
            GOTO      BROUND32

            CPFSGT    BEXP              ; do another byte shift if BEXP >= 8
            GOTO      BNIB32A
            SUBWF     AEXP              ; BEXP = BEXP - 7
            RLCF      AARG+B2           ; rotate next bit for rounding
            MOVFP     AARG+B1,AARG+B2
            CLRF      AARG+B1
            DCFSNZ    BEXP              ; BEXP = BEXP - 1
            GOTO      BROUND32

            CPFSGT    BEXP              ; if BEXP still >= 8, then
            GOTO      BNIB32B           ; AARG = 0 relative to BARG

            MOVFP     SIGN,AARG+B0      ; return BARG
            MOVFP     BARG+B1,AARG+B1
            MOVFP     BARG+B2,AARG+B2
            RETLW     0x00

BNIB32B     MOVLW     3                 ; do nibbleshift if BEXP >= 4
            CPFSGT    BEXP
            GOTO      BLOOP32B
            SUBWF     BEXP              ; BEXP = BEXP -3

            SWAPF     AARG+B2,W
            MOVPF     WREG,ACC+B5       ; save for rounding
            ANDLW     0x0F
            MOVPF     WREG,AARG+B2
            RLCF      ACC+B5            ; rotate next bit for rounding

            DCFSNZ    BEXP              ; BEXP = BEXP - 1
            GOTO      BROUND32          ; aligned if BEXP = 0

BLOOP32B    BCF       _C                ; right shift by BEXP
            RRCF      AARG+B2
            DCFSNZ    BEXP
            GOTO      BROUND32          ; aligned if BEXP = 0
            BCF       _C
            RRCF      AARG+B2
            DCFSNZ    BEXP
            GOTO      BROUND32          ; aligned if BEXP = 0
            BCF       _C                ; at most 3 right shifts are
            RRCF      AARG+B2           ; possible
            GOTO      BROUND32

BNIB32A     MOVLW     3                 ; do nibbleshift if BEXP >= 4
            CPFSGT    BEXP
            GOTO      BLOOP32A
            SUBWF     BEXP              ; BEXP = BEXP -3
```


© 1994 Microchip Technology Inc.

```
                SWAPF       AARG+B2,W
                MOVPF       WREG,ACC+B5         ; save for rounding
                ANDLW       0x0F
                MOVPF       WREG,AARG+B2

                SWAPF       AARG+B1,W
                ANDLW       0xF0
                ADDWF       AARG+B2

                SWAPF       AARG+B1,W
                ANDLW       0x0F
                MOVPF       WREG,AARG+B1

                RLCF        ACC+B5              ; rotate next bit for rounding
                DCFSNZ      BEXP                ; BEXP = BEXP - 1
                GOTO        BROUND32            ; aligned if BEXP = 0

BLOOP32A        BCF         _C                  ; right shift by BEXP
                RRCF        AARG+B1
                RRCF        AARG+B2
                DCFSNZ      BEXP
                GOTO        BROUND32            ; aligned if BEXP = 0
                BCF         _C
                RRCF        AARG+B1
                RRCF        AARG+B2
                DCFSNZ      BEXP
                GOTO        BROUND32            ; aligned if BEXP = 0
                BCF         _C                  ; at most 3 right shifts are
                RRCF        AARG+B1             ; possible
                RRCF        AARG+B2
                GOTO        BROUND32

BNIB32          MOVLW       3                   ; do nibbleshift if BEXP >= 4
                CPFSGT      BEXP
                GOTO        BLOOP32
                SUBWF       BEXP                ; BEXP = BEXP -3

                SWAPF       AARG+B2,W
                MOVPF       WREG,ACC+B5         ; save for rounding
                ANDLW       0x0F
                MOVPF       WREG,AARG+B2

                SWAPF       AARG+B1,W
                ANDLW       0xF0
                ADDWF       AARG+B2

                SWAPF       AARG+B1,W
                ANDLW       0x0F
                MOVPF       WREG,AARG+B1

                SWAPF       AARG+B0,W
                ANDLW       0xF0
                ADDWF       AARG+B1

                SWAPF       AARG+B0,W
                ANDLW       0x0F
                MOVPF       WREG,AARG+B0

                RLCF        ACC+B5              ; rotate next bit for rounding

                DCFSNZ      BEXP                ; BEXP = BEXP - 1
                GOTO        BROUND32            ; aligned if BEXP = 0

BLOOP32         BCF         _C                  ; right shift by BEXP
                RRCF        AARG+B0
                RRCF        AARG+B1
                RRCF        AARG+B2
```


© 1994 Microchip Technology Inc.

```
                DCFSNZ          BEXP
                GOTO            BROUND32        ; aligned if BEXP = 0
                BCF             _C
                RRCF            AARG+B0
                RRCF            AARG+B1
                RRCF            AARG+B2
                DCFSNZ          BEXP
                GOTO            BROUND32        ; aligned if BEXP = 0
                BCF             _C              ; at most 3 right shifts are
                RRCF            AARG+B0         ; possible
                RRCF            AARG+B1
                RRCF            AARG+B2

BROUND32        BTFSC           FPFLAGS,RND
                BTFSS           AARG+B2,LSB
                GOTO            BLIGNED32
                CLRF            WREG
                ADDWFC          AARG+B2         ; add carry for rounding
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0

                BTFSS           _C              ; has rounding caused carryout?
                GOTO            BLIGNED32
                RRCF            AARG+B0         ; if so, right shift
                RRCF            AARG+B1
                RRCF            AARG+B2
                INFSNZ          EXP             ; test for floating point overflow
                GOTO            SETFOV32

BLIGNED32       BTFSS           TEMP,MSB        ; negate if signs opposite
                GOTO            AOK32
                CLRF            WREG
                COMF            AARG+B2
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                GOTO            AOK32

USEA32          MOVFP           AARG+B0,SIGN    ; save sign in SIGN
                BSF             AARG+B0,MSB     ; make MSB's explicit
                BSF             BARG+B0,MSB

                MOVFP           BEXP,WREG       ; compute shift count in BEXP
                SUBWF           AEXP,W
                MOVPF           WREG,BEXP
                BTFSC           _Z
                GOTO            ALIGNED32

                MOVLW           7
                CPFSGT          BEXP            ; do byte shift if BEXP >= 8
                GOTO            ANIB32
                SUBWF           BEXP            ; BEXP = BEXP - 7
                RLCF            BARG+B2         ; rotate next bit for rounding
                MOVFP           BARG+B1,WREG
                MOVPF           WREG,BARG+B2
                MOVFP           BARG+B0,WREG
                MOVPF           WREG,BARG+B1
                CLRF            BARG+B0
                DCFSNZ          BEXP            ; BEXP = BEXP - 1
                GOTO            AROUND32

                CPFSGT          BEXP            ; do another byte shift if BEXP >= 8
                GOTO            ANIB32A
                SUBWF           BEXP            ; BEXP = BEXP - 7
                RLCF            BARG+B2         ; rotate next bit for rounding
                MOVFP           BARG+B1,WREG
                MOVPF           WREG,BARG+B2
```


© 1994 Microchip Technology Inc.

```
                CLRF            BARG+B1
                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32

                CPFSGT          BEXP                    ; if BEXP still >= 8, then
                GOTO            ANIB32B                 ; BARG = 0 relative to AARG

                MOVFP           SIGN,AARG+B0            ; return AARG
                RETLW           0x00

ANIB32B         MOVLW           3                       ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32B
                SUBWF           BEXP                    ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2
                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32                ; aligned if BEXP = 0

ALOOP32B        BCF             _C                      ; right shift by BEXP
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C                      ; at most 3 right shifts are
                RRCF            BARG+B2                 ; possible
                GOTO            AROUND32

ANIB32A         MOVLW           3                       ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32A
                SUBWF           BEXP                    ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5             ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0xF0
                ADDWF           BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1

                RLCF            ACC+B5                  ; rotate next bit for rounding

                DCFSNZ          BEXP                    ; BEXP = BEXP - 1
                GOTO            AROUND32                ; aligned if BEXP = 0

ALOOP32A        BCF             _C                      ; right shift by BEXP
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32                ; aligned if BEXP = 0
```


© 1994 Microchip Technology Inc.

```
                BCF             _C              ; at most 3 right shifts are
                RRCF            BARG+B1         ; possible
                RRCF            BARG+B2
                GOTO            AROUND32

ANIB32          MOVLW           3               ; do nibbleshift if BEXP >= 4
                CPFSGT          BEXP
                GOTO            ALOOP32
                SUBWF           BEXP            ; BEXP = BEXP -3

                SWAPF           BARG+B2,W
                MOVPF           WREG,ACC+B5     ; save for rounding
                ANDLW           0x0F
                MOVPF           WREG,BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0xF0
                ADDWF           BARG+B2

                SWAPF           BARG+B1,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B1

                SWAPF           BARG+B0,W
                ANDLW           0xF0
                ADDWF           BARG+B1

                SWAPF           BARG+B0,W
                ANDLW           0x0F
                MOVPF           WREG,BARG+B0

                RLCF            ACC+B5          ; rotate next bit for rounding

                DCFSNZ          BEXP            ; BEXP = BEXP - 1
                GOTO            AROUND32        ; aligned if BEXP = 0

ALOOP32         BCF             _C              ; right shift by BEXP
                RRCF            BARG+B0
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32        ; aligned if BEXP = 0
                BCF             _C
                RRCF            BARG+B0
                RRCF            BARG+B1
                RRCF            BARG+B2
                DCFSNZ          BEXP
                GOTO            AROUND32        ; aligned if BEXP = 0
                BCF             _C              ; at most 3 right shifts are
                RRCF            BARG+B0         ; possible
                RRCF            BARG+B1
                RRCF            BARG+B2

AROUND32        BTFSC           FPFLAGS,RND
                BTFSS           BARG+B2,LSB
                GOTO            ALIGNED32
                CLRF            WREG
                ADDWFC          BARG+B2         ; add carry for rounding
                ADDWFC          BARG+B1
                ADDWFC          BARG+B0

                BTFSS           _C              ; has rounding caused carryout?
                GOTO            ALIGNED32
                RRCF            BARG+B0         ; if so, right shift
                RRCF            BARG+B1
                RRCF            BARG+B2
                INFSNZ          EXP             ; test for floating point overflow
                GOTO            SETFOV32
```

5

```
ALIGNED32       BTFSS           TEMP,MSB        ; negate if signs opposite
                GOTO            AOK32
                CLRF            WREG
                COMF            BARG+B2
                COMF            BARG+B1
                COMF            BARG+B0
                INCF            BARG+B2
                ADDWFC          BARG+B1
                ADDWFC          BARG+B0

AOK32           MOVFP           BARG+B2,WREG    ; add
                ADDWF           AARG+B2
                MOVFP           BARG+B1,WREG
                ADDWFC          AARG+B1
                MOVFP           BARG+B0,WREG
                ADDWFC          AARG+B0

                BTFSC           TEMP,MSB
                GOTO            ACOMP32
                BTFSS           _C
                GOTO            FIXSIGN32

                RRCF            AARG+B0         ; shift right and increment EXP
                RRCF            AARG+B1
                RRCF            AARG+B2
                INCFSZ          AEXP
                GOTO            FIXSIGN32
                GOTO            SETFOV32        ; set floating point overflow flag

ACOMP32         BTFSC           _C
                GOTO            NRM32           ; normalize and fix sign

                CLRF            WREG            ; negate, toggle sign bit and
                COMF            AARG+B2         ; then normalize
                COMF            AARG+B1
                COMF            AARG+B0
                INCF            AARG+B2
                ADDWFC          AARG+B1
                ADDWFC          AARG+B0
                BTG             SIGN,MSB
                GOTO            NRM32

                END
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN526

PIC16C5X / 16CXX Math Utility Routines

INTRODUCTION

This application note provides some utility math routines for Microchip's PIC16C5X and PIC16CXX series of 8-bit microcontrollers. The following math routines are provided:

- 8 x 8 unsigned multiply
- 16 x 16 double precision multiply
- Fixed Point Division (see Table 3)
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- BCD to binary conversion routines
- binary to BCD conversion routines
- BCD addition
- BCD subtraction
- square root

These are written in native assembly language and the listing files are provided. They are also available on a disk (MS-DOS®). All the routines provided can be called as subroutines. Most of the routines have two different versions: one optimized for speed and the other optimized for code size. The calling sequence of each routine is explained at the beginning of each listing file.

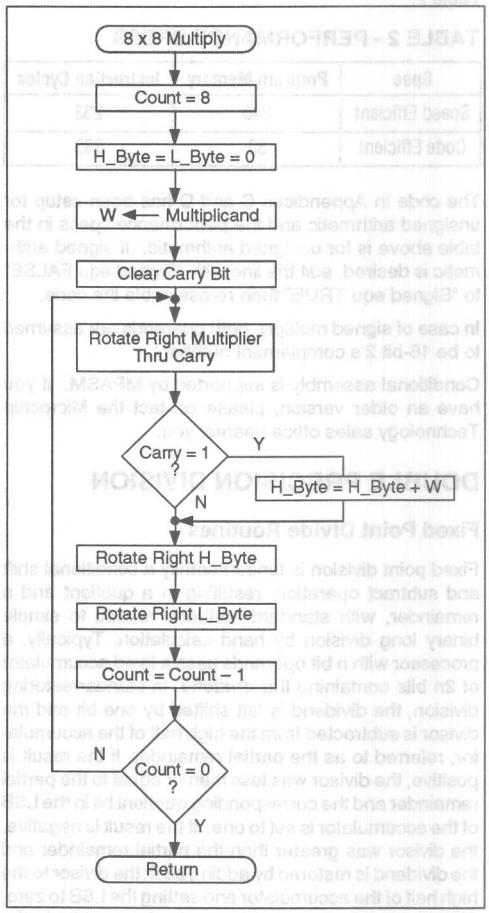
SINGLE PRECISION UNSIGNED MULTIPLICATION (8 X 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices A and B. The performance specs for the routines are shown in Table 1.

TABLE 1 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	35	37
Code Efficient	16	71

FIGURE 1 - FLOW CHART FOR UN SIGNED 8 X 8 MULTIPLY



5

© 1994 Microchip Technology Inc.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of two 16-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic are supported. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices C and D. The performance specs for the routines are shown in Table 2.

TABLE 2 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	240	233
Code Efficient	33	333

The code in Appendices C and D has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "Signed equ FALSE" to "Signed equ TRUE" then re-assemble the code.

In case of signed multiply, both operands are assumed to be 16-bit 2's complement numbers.

Conditional assembly is supported by MPASM. If you have an older version, please contact the Microchip Technology sales office nearest you.

DOUBLE PRECISION DIVISION

Fixed Point Divide Routines

Fixed point division is fundamentally a conditional shift and subtract operation, resulting in a quotient and a remainder, with standard methods related to simple binary long division by hand calculation. Typically, a processor with n bit operands uses a fixed accumulator of 2n bits containing the dividend. In standarrestoring division, the dividend is left shifted by one bit and the divisor is subtracted from the high half of the accumulator, referred to as the partial remainder. If the result is positive, the divisor was less than or equal to the partial remainder and the corresponding quotient bit in the LSB of the accumulator is set to one. If the result is negative, the divisor was greater than the partial remainder and the dividend is restored by adding back the divisor to the high half of the accumulator and setting the LSB to zero. This process is repeated for each of the n bits in the divisor, resulting in an n bit quotient in the low half of the accumulator and the n bit remainder in the high half, and requiring n subtractions and on average n/2 additions [1].

Nonrestoring division, requiring a total of at most n+1 subtractions and additions, offers potential for speed improvement by allowing a negative partial remainder during the calculation with a final addition of the divisor if the final remainder is negative. After the first left shift of the dividend, the divisor is subtracted and the corresponding quotient bit as well as the next add or subtract operation is determined by the carry bit [1].

Unfortunately, no simple methods exist for performing two's complement binary division, thereby requiring negate operations during a preprocessing phase. It is important to note that with the dividend initially loaded into the accumulator, an overflow of the final quotient will result if the high half of the dividend is greater than or equal to the divisor [1], indicating that the n bit range of the quotient will be exceeded.

Because of the inherent byte structure of the PIC16/17 family of microcontrollers, a more creative and efficient implementation of the above algorithms is possible. In what follows, the partial remainder is initialized at zero and is separated from the dividend, thereby avoiding any alignment logic overhead and yielding a quotient with the same number of bits as the dividend and a remainder with the same number as the divisor. Furthermore, the routines are named in the format FXDxxyyz, where xx is the number of bits in the dividend, yy is the number of bits in the divisor, and z indicates a signed or unsigned routine. Macros are used for the core sections of each routine, thereby permitting simple switching between restoring and nonrestoring methods. The signed macros are exclusively a variation of the nonrestoring method, taking advantage of the zero MSB of the arguments after the preprocessing negation. Both restoring and nonrestoring macros are included for the unsigned case, with selection based on best worst case or best average performance as desired. For example, the unsigned macros exhibit the following performance data:

		32/16	16/16	16/8
restore	**max.**	561	240	193
	ave.	481	208	173
nonrestore	**max.**	481	240	190
	ave.	466	233	183

This demonstrates that while the nonrestoring algorithm is preferred for the 32/16 case, the restoring method is preferred for the 16/16 case, with the choice for the 16/8 case a function of user requirements. These optimization complications are a result of trade-offs between the number of instructions required for the restore operations verses the added logic requirements. Finally, additional routines with tacit MSB equal to zero in each argument are included, yielding significant speed improvement. These routines can also be called in the signed case when the arguments are known to be positive for a small benefit.

Routines

It is useful to note that the additional routines FXD3115U, FXD1515U, and FXD1507U can be called in a signed divide application in the special case where AARG > 0 and BARG > 0, thereby offering some improvement in performance.

© 1994 Microchip Technology Inc.

TABLE 3: FIXED POINT DIVIDE ROUTINES

Routine	Clocks	Function
FXD3216S	414	32 bit/16 bit -> 32.16 signed fixed point divide
FXD3216U	485	32 bit/16 bit -> 32.16 unsigned fixed point divide
FXD3215U	390	32 bit/15 bit -> 32.15 unsigned fixed point divide
FXD3115U	383	31 bit/15 bit -> 31.15 unsigned fixed point divide
FXD1616S	214	16 bit/16 bit -> 16.16 signed fixed point divide
FXD1616U	244	16 bit/16 bit -> 16.16 unsigned fixed point divide
FXD1615U	197	16 bit/15 bit -> 16.15 unsigned fixed point divide
FXD1515U	191	15 bit/15 bit -> 15.15 unsigned fixed point divide
FXD1608S	146	16 bit/08 bit -> 16.08 signed fixed point divide
FXD1608U	196	16 bit/08 bit -> 16.08 unsigned fixed point divide
FXD1607U	130	16 bit/07 bit -> 16.07 unsigned fixed point divide
FXD1507U	125	15 bit/07 bit -> 15.07 unsigned fixed point divide

Data RAM Requirements

The following contiguous data ram locations are used by the fixed point divide routines:

ACC+B0	=	AARG+B0	AARG and ACC
ACC+B1	=	AARG+B1	
ACC+B2	=	AARG+B2	
ACC+B3	=	AARG+B3	
ACC+B4	=	REM+B0	remainder
ACC+B5	=	REM+B1	
SIGN			sign in MSB
BARG+B0			BARG
BARG+B1			
TEMP+B0			temporary storage
TEMP+B1			

where Bx = x.

References

1. Cavanagh, J.J.F., "Digital Computer Arithmetic," McGraw-Hill,1984.
2. Hwang, K., "Computer Arithmetic," John Wiley & Sons, 1979.
3. Scott, N.R., "Computer Number Systems & Arithmetic," Prentice Hall, 1985.

5

TABLE 4: PIC16CXX FIXED POINT DIVIDE PERFORMANCE DATA

Routine	Max Cycles	Min. Cycles	PM	DM
16 / 8 Signed	146	135	146	5
16 / 8 Unsigned	196	156	195	4
16 / 7 Unsigned	130	130	129	4
15 / 7 Unsigned	125	125	124	4
16 / 16 Unsigned	214	187	241	7
16 / 16 Unsigned	244	180	243	6
16 / 15 Unsigned	197	182	216	6
16 / 15 Unsigned	191	177	218	6
32 / 16 Unsigned	414	363	476	9
32 / 16 Unsigned	485	459	608	9
32 / 15 Unsigned	390	359	451	8
31 / 15 Unsigned	383	353	442	8

© 1994 Microchip Technology Inc.

DOUBLE PRECISION ADDITION & SUBTRACTION

This routine adds or subtracts two 16-bit numbers and produces a 16-bit result. This routine is used by other double precision routines. The listing of these routines is given in Appendix E. The performance specs for the routines are shown below:

TABLE 5 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Addition	7	8
Subtraction	14	17

BCD TO BINARY CONVERSION

This routine converts a five digit BCD number to a 16-bit binary number. The listing of this routine is given in Appendix F. The performance spec for the routine is shown below:

TABLE 6 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD to Binary	30	121

© 1994 Microchip Technology Inc.

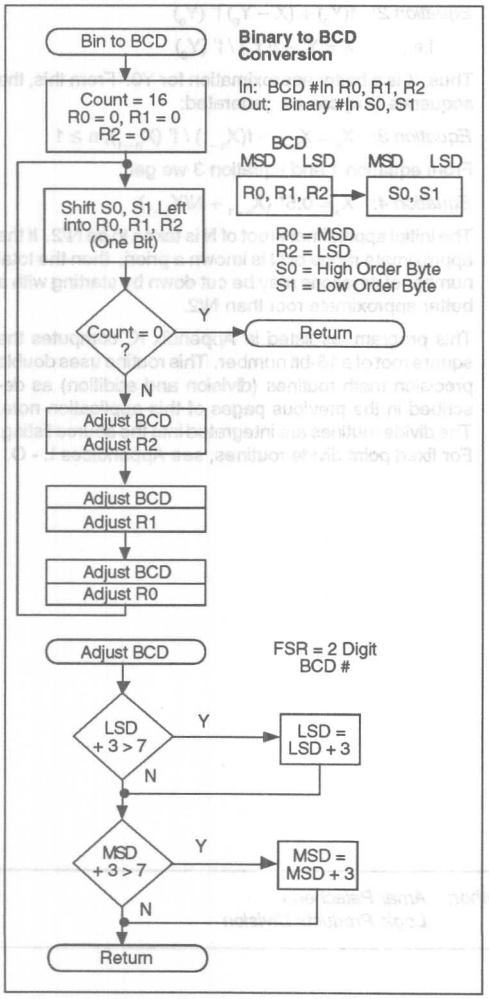
BINARY TO BCD CONVERSION

Two routines are provided: one routine converts a 16-bit binary number to a five-digit BCD number and the other routine converts an 8-bit binary number to a two-digit BCD number. The listing of these routines are given in Appendices G and H. The performance specs for the routines are shown below:

TABLE 7 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Binary (8-Bit) to BCD	10	81 (Worst Case)
Binary (16-Bit) to BCD	30	719

FIGURE 2 - FLOW CHART FOR BINARY TO BCD CONVERSION



BCD ADDITION & SUBTRACTION

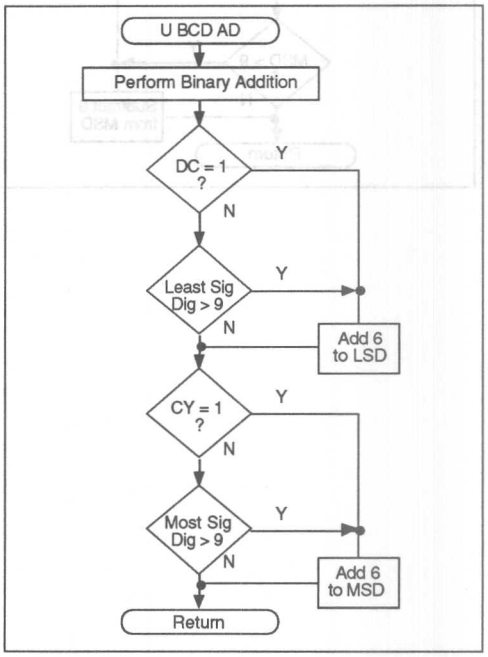
These two routines perform a two-digit unsigned BCD addition and subtraction. The results are the sum (or difference) in one File register and with a overflow carry-bit in another File register. The performance specs for the routines are shown below:

TABLE 8 - PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD Addition	29	23 (Worst Case)
BCD Subtraction	31	21 (Worst Case)

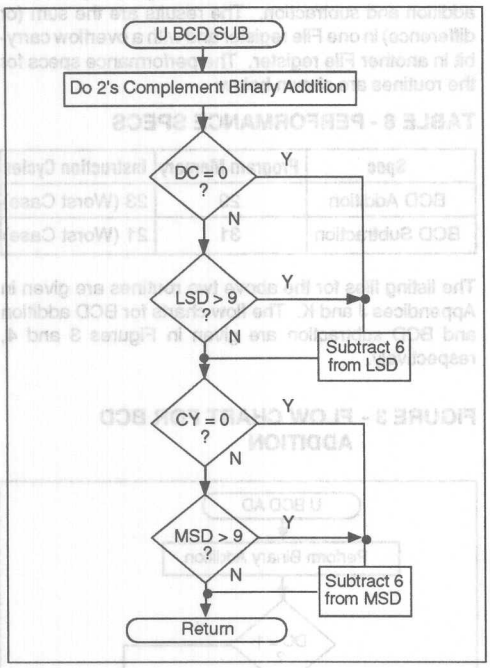
The listing files for the above two routines are given in Appendices J and K. The flow charts for BCD addition and BCD subtraction are given in Figures 3 and 4, respectively.

FIGURE 3 - FLOW CHART FOR BCD ADDITION



5

FIGURE 4 - FLOW CHART FOR BCD SUBTRACTION



SQUARE ROOT

Often in many applications one needs to find the square root of a number. Of many numerical methods to find the square root of a number, the Newton-Raphson method is very attractive because of its fast convergence rate. In this method the square root of a number, "N", is obtained from the approximate solution of:

$$f(Y) = Y^2 - N = 0$$

The function "f(Y)" can be expanded about Y_0 using first order Taylor polynomial expansion as:

Equation 1: $f(Y) = f(Y_0) + (Y - Y_0) f'(Y_0) + (Y - Y_0)^2 f''(Y_0) / 2! + \ldots$

If X is a root of f(Y), then f(X) = 0:

$$f(X) = f(Y_0) + (x - Y_0) f'(Y_0) + (X - Y_0)^2 f''(Y_0) / 2! + \ldots = 0$$

If Y0 is an approximate root of f(Y), then higher order terms are negligible. Therefore:

Equation 2: $f(Y_0) + (X - Y_0) f'(Y_0)$

i.e., $X = Y_0 - f(Y_0) / f'(Y_0)$

Thus, X is a better approximation for Y0. From this, the sequence {Xn} can be generated:

Equation 3: $X_n = X_{n-1} - f(X_{n-1}) / f'(X_{n-1}), n \geq 1$

From equation 1 and equation 3 we get,

Equation 4: $X_n = 0.5^* \{X_{n-1} + N/X_{n-1}\}$

The initial approximate root of N is taken to be N/2. If the approximate range of N is known a priori, then the total number of iterations may be cut down by starting with a better approximate root than N/2.

This program, as listed in Appendix K, computes the square root of a 16-bit number. This routine uses double precision math routines (division and addition) as described in the previous pages of this application note. The divide routines are integrated into the source listing. For fixed point divide routines, see Appendices L - O.

Author: Amar Palacherla
Logic Products Division

© 1994 Microchip Technology Inc.

APPENDIX A

```
MPASM B0.54                                                    PAGE  1


                ;*****************************************************************
                ;                   8x8 Software Multiplier
                ;                ( Code Efficient : Looped Code )
                ;*****************************************************************
                ;
                ;   The 16 bit result is stored in 2 bytes
                ;
                ; Before calling the subroutine " mpy ", the multiplier should
                ; be loaded in location " mulplr ", and the multiplicand in
                ; " mulcnd " . The 16 bit result is stored in locations
                ; H_byte & L_byte.
                ;
                ;       Performance :
                ;                       Program Memory  :  15 locations
                ;                       # of cycles     :  71
                ;                       Scratch RAM     :   0 locations
                ;
                ;  This routine is optimized for code efficiency ( looped code )
                ;  For time efficiency code refer to "mult8x8F.asm" ( straight line code )
                ;*****************************************************************
                ;
                        LIST    P=16C54
0009            mulcnd  equ     09      ; 8 bit multiplicand
0010            mulplr  equ     10      ; 8 bit multiplier
0012            H_byte  equ     12      ; High byte of the 16 bit result
0013            L_byte  equ     13      ; Low byte of the 16 bit result
0014            count   equ     14      ; loop counter
                ;
                ;
                  include       "mpreg.h"
                ;*************************       PIC16C5X Header ************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
```

5

© 1994 Microchip Technology Inc.

```
                        ;
0001                    Same    equ     1h
                        ;
0000                    LSB     equ     0h
0007                    MSB     equ     7h
                        ;
0001                    TRUE    equ     1h
0001                    YES     equ     1h
0000                    FALSE   equ     0h
0000                    NO      equ     0h
                        ;
                        ;*****************************************************************************


                        ;
                        ; ****************************        Begin Multiplier Routine
0000 0072               mpy_S   clrf    H_byte
0001 0073                       clrf    L_byte
0002 0C08                       movlw   8
0003 0034                       movwf   count
0004 0209                       movf    mulcnd,w
0005 0403                       bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
0006 0330               loop    rrf     mulplr
0007 0603                       btfsc   STATUS,CARRY
0008 01F2                       addwf   H_byte,Same
0009 0332                       rrf     H_byte,Same
000A 0333                       rrf     L_byte,Same
000B 02F4                       decfsz  count
000C 0A06                       goto    loop
                        ;
000D 0800                       retlw   0
                        ;
                        ;*****************************************************************
                        ;               Test Program
                        ;*****************************************************************
000E 0CFF               main    movlw   0FF
000F 0030                       movwf   mulplr          ; multiplier (in mulplr) = 0FF
0010 0CFF                       movlw   0FF             ; multiplicand(W Reg )   = 0FF
0011 0029                       movwf   mulcnd
                        ;
0012 0900                       call    mpy_S           ; The result 0FF*0FF = FE01 is in locations
                        ;                               ; H_byte & L_byte
                        ;
0013 0A13               self    goto    self
                        ;
                                org     01FF
01FF 0A0E                       goto    main
                        ;
                                END


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX B

```
MPASM B0.54                                                    PAGE  1


                ;*****************************************************************
                ;                    8x8 Software Multiplier
                ;                ( Fast Version : Straight Line Code )
                ;*****************************************************************
                ;
                ;   The 16 bit result is stored in 2 bytes
                ;
                ; Before calling the subroutine " mpy ", the multiplier should
                ; be loaded in location " mulplr ", and the multiplicand in
                ; " mulcnd " . The 16 bit result is stored in locations
                ; H_byte & L_byte.
                ;
                ;       Performance :
                ;                       Program Memory  :  35 locations
                ;                       # of cycles     :  37
                ;                       Scratch RAM     :   0 locations
                ;
                ; This routine is optimized for speed efficiency ( straight line code )
                ;  For code efficiency, refer to "mult8x8S.asm" ( looped code )
                ;*****************************************************************
                ;
                        LIST    P=16C54
0009            mulcnd  equ     09      ; 8 bit multiplicand
0010            mulplr  equ     10      ; 8 bit multiplier
0012            H_byte  equ     12      ; High byte of the 16 bit result
0013            L_byte  equ     13      ; Low byte of the 16 bit result
                ;
                ;
                  include         "picreg.h"
                ;*************************       PIC16C5X Header ************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
                ;
0001            Same    equ     1h
                ;
```

5

© 1994 Microchip Technology Inc.

```
0000                    LSB     equ     0h
0007                    MSB     equ     7h
                        ;
0001                    TRUE    equ     1h
0001                    YES     equ     1h
0000                    FALSE   equ     0h
0000                    NO      equ     0h
                        ;
                        ;*****************************************************************************
                        ;
                        ;****   Define a macro for adding & right shifting  **
                        ;
                        mult    MACRO   bit             ; Begin macro
                          btfsc   mulplr,bit
                          addwf   H_byte,Same
                          rrf     H_byte,Same
                          rrf     L_byte,Same
                          ENDM                          ; End of macro
                        ;
                        ; ****************************        Begin Multiplier Routine
0000 0072               mpy_F   clrf    H_byte
0001 0073                       clrf    L_byte
0002 0209                       movf    mulcnd,w        ; move the multiplicand to W reg.
0003 0403                       bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
                        mult    0
0004 0610                       btfsc   mulplr,0
0005 01F2                       addwf   H_byte,Same
0006 0332                       rrf     H_byte,Same
0007 0333                       rrf     L_byte,Same

                        mult    1
0008 0630                       btfsc   mulplr,1
0009 01F2                       addwf   H_byte,Same
000A 0332                       rrf     H_byte,Same
000B 0333                       rrf     L_byte,Same

                        mult    2
000C 0650                       btfsc   mulplr,2
000D 01F2                       addwf   H_byte,Same
000E 0332                       rrf     H_byte,Same
000F 0333                       rrf     L_byte,Same

                        mult    3
0010 0670                       btfsc   mulplr,3
0011 01F2                       addwf   H_byte,Same
0012 0332                       rrf     H_byte,Same
0013 0333                       rrf     L_byte,Same

                        mult    4
0014 0690                       btfsc   mulplr,4
0015 01F2                       addwf   H_byte,Same
0016 0332                       rrf     H_byte,Same
0017 0333                       rrf     L_byte,Same
                        mult    5
0018 06B0                       btfsc   mulplr,5
0019 01F2                       addwf   H_byte,Same
001A 0332                       rrf     H_byte,Same
001B 0333                       rrf     L_byte,Same

                        mult    6
001C 06D0                       btfsc   mulplr,6
001D 01F2                       addwf   H_byte,Same
001E 0332                       rrf     H_byte,Same
001F 0333                       rrf     L_byte,Same
```


© 1994 Microchip Technology Inc.

```
                        mult    7
0020 06F0                       btfsc   mulplr,7
0021 01F2                       addwf   H_byte,Same
0022 0332                       rrf     H_byte,Same
0023 0333                       rrf     L_byte,Same

                        ;
0024 0800                       retlw   0
                        ;
                        ;*****************************************************************
                        ;               Test Program
                        ;*****************************************************************
0025 0CFF               main    movlw   0FF
0026 0030                       movwf   mulplr          ; multiplier (in mulplr)      = 0FF
0027 0CFF                       movlw   0FF
0028 0029                       movwf   mulcnd          ; multiplicand(in mulcnd )    = 0FF
                        ;
0029 0900                       call    mpy_F           ; The result 0FF*0FF = FE01 is in locations
                        ;                               ; H_byte & L_byte
                        ;
002A 0A2A               self    goto    self
                        ;
                                org     01FF
01FF 0A25                       goto    main
                        ;
                                END


Errors   :     0
Warnings :     0
```

5

APPENDIX C: DOUBLE PRECISION MULTIPLICATION LISTING (LOOPED)

```
MPASM B0.54                                              PAGE  1

                    ;*****************************************************************
                    ;                   Double Precision Multiplication
                    ;
                    ;               ( Optimized for Code Size : Looped Code )
                    ;
                    ;*****************************************************************;
                    ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
                    ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    ;       (c) CALL D_mpy
                    ;       (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
                    ;
                    ;   Performance :
                    ;                   Program Memory  :       033
                    ;                   Clock Cycles    :       333
                    ;
                    ;       Note : The above timing is the worst case timing, when the
                    ;              register ACCb = FFFF. The speed may be improved if
                    ;              the register ACCb contains a number ( out of the two
                    ;              numbers ) with less number of 1s.
                    ;              The performance specs are for Unsigned arithmetic ( i.e,
                    ;              with "SIGNED equ  FALSE ").
                    ;
                    ;              The performance specs are for Unsigned arithmetic ( i.e,
                    ;              with "SIGNED equ  FALSE ").
                    ;
                    ;*****************************************************************;
                    ;
                            LIST    P=16C54
0010                ACCaLO  equ     10
0011                ACCaHI  equ     11
0012                ACCbLO  equ     12
0013                ACCbHI  equ     13
0014                ACCcLO  equ     14
0015                ACCcHI  equ     15
0016                ACCdLO  equ     16
0017                ACCdHI  equ     17
0018                temp    equ     18
0019                sign    equ     19
                    ;
                      include "picreg.h"
                    ;*************************        PIC16C5X Header ************************
01FF                PIC54   equ     1FFH    ; Define Reset Vectors
01FF                PIC55   equ     1FFH
03FF                PIC56   equ     3FFH
07FF                PIC57   equ     7FFH
                    ;
0001                RTCC    equ     1h
0002                PC      equ     2h
0003                STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004                FSR     equ     4h
                    ;
0005                Port_A  equ     5h
0006                Port_B  equ     6h      ; I/O Port Assignments
0007                Port_C  equ     7h
                    ;
                    ;*****************************************************************************
                    ;
                                            ; STATUS REG. Bits
0000                CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000                C       equ     0h
0001                DCARRY  equ     1h
0001                DC      equ     1h
0002                Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002                Z       equ     2h
```


© 1994 Microchip Technology Inc.

```
0003              P_DOWN  equ     3h
0003              PD      equ     3h
0004              T_OUT   equ     4h
0004              TO      equ     4h
0005              PA0     equ     5h
0006              PA1     equ     6h
0007              PA2     equ     7h
                  ;
0001              Same    equ     1h
                  ;
0000              LSB     equ     0h
0007              MSB     equ     7h
                  ;
0001              TRUE    equ     1h
0001              YES     equ     1h
0000              FALSE   equ     0h
0000              NO      equ     0h
                  ;
                  ;*****************************************************************************


                        org     0
                  ;*****************************************************************
0001              SIGNED  equ     TRUE            ; Set This To 'TRUE' if the routines
                  ;                               ; for Multiplication & Division needs
                  ;                               ; to be assembled as Signed Integer
                  ;                               ; Routines. If 'FALSE' the above two
                  ;                               ; routines ( D_mpy & D_div ) use
                  ;                               ; unsigned arithmetic.
                  ;*****************************************************************
                  ;         Double Precision Subtraction ( ACCb - ACCa -> ACCb )
                  ;
0000 0210         D_add   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0001 01F2                 addwf   ACCbLO          ;add lsb
0002 0603                 btfsc   STATUS,CARRY    ;add in carry
0003 02B3                 incf    ACCbHI
0004 0211                 movf    ACCaHI,w
0005 01F3                 addwf   ACCbHI          ;add msb
0006 0800                 retlw   0
                  ;*****************************************************************
                  ;               Double Precision Multiply ( 16x16 -> 32 )
                  ;          ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                  ; in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                  ;
                  D_mpyS                          ;results in ACCb(16 msb's) and ACCc(16
                  ;
                      IF   SIGNED
0007 0930               CALL    S_SIGN
                      ENDIF
                  ;
0008 0921                 call    setup
0009 0337         mloop   rrf     ACCdHI          ;rotate d right
000A 0336                 rrf     ACCdLO
000B 0603                 btfsc   STATUS,CARRY    ;need to add?
000C 0900                 call    D_add
000D 0333                 rrf     ACCbHI
000E 0332                 rrf     ACCbLO
000F 0335                 rrf     ACCcHI
0010 0334                 rrf     ACCcLO
0011 02F8                 decfsz  temp            ;loop until all bits checked
0012 0A09                 goto    mloop
```

5

© 1994 Microchip Technology Inc.

```
                    ;
                       IF   SIGNED
0013 07F9                   btfss   sign,MSB
0014 0800                   retlw   0
0015 0274                   comf    ACCcLO          ; negate ACCa ( -ACCa -> ACCa )
0016 02B4                   incf    ACCcLO
0017 0643                   btfsc   STATUS,Z_bit
0018 00F5                   decf    ACCcHI
0019 0275                   comf    ACCcHI
001A 0643                   btfsc   STATUS,Z_bit
001B 0272                neg_B  comf    ACCbLO          ; negate ACCb
001C 02B2                   incf    ACCbLO
001D 0643                   btfsc   STATUS,Z_bit
001E 00F3                   decf    ACCbHI
001F 0273                   comf    ACCbHI
0020 0800                   retlw   0
                          ELSE
                              retlw   0
                          ENDIF
                    ;
                    ;*****************************************************************
                    ;
0021 0C10           setup   movlw   .16             ; for 16 shifts
0022 0038                   movwf   temp
0023 0213                   movf    ACCbHI,w        ;move ACCb to ACCd
0024 0037                   movwf   ACCdHI
0025 0212                   movf    ACCbLO,w
0026 0036                   movwf   ACCdLO
0027 0073                   clrf    ACCbHI
0028 0072                   clrf    ACCbLO
0029 0800                   retlw   0
                    ;
                    ;*****************************************************************
                    ;
002A 0270            neg_A   comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
002B 02B0                    incf    ACCaLO
002C 0643                    btfsc   STATUS,Z_bit
002D 00F1                    decf    ACCaHI
002E 0271                    comf    ACCaHI
002F 0800                    retlw   0
                    ;
                    ;*****************************************************************
                    ;  Assemble this section only if Signed Arithmetic Needed
                    ;
                       IF    SIGNED
                    ;
0030 0211           S_SIGN  movf    ACCaHI,W
0031 0193                   xorwf   ACCbHI,W
0032 0039                   movwf   sign
0033 07F3                   btfss   ACCbHI,MSB      ; if MSB set go & negate ACCb
0034 0A3A                   goto    chek_A
                    ;
0035 0272                   comf    ACCbLO          ; negate ACCb
0036 02B2                   incf    ACCbLO
0037 0643                   btfsc   STATUS,Z_bit
0038 00F3                   decf    ACCbHI
0039 0273                   comf    ACCbHI
                    ;
003A 07F1           chek_A  btfss   ACCaHI,MSB      ; if MSB set go & negate ACCa
003B 0800                   retlw   0
003C 0A2A                   goto    neg_A
                    ;
                        ENDIF
```


© 1994 Microchip Technology Inc.

```
                        ;
                        ;*****************************************************************
                        ;                           Test Program
                        ;*****************************************************************
                        ;    Load constant values to ACCa & ACCb for testing
                        ;
003D 0C01               main    movlw   1
003E 0031                       movwf   ACCaHI
003F 0CFF                       movlw   0FF             ; loads ACCa = 01FF
0040 0030                       movwf   ACCaLO
                        ;
0041 0C7F                       movlw   07F
0042 0033                       movwf   ACCbHI
0043 0CFF                       movlw   0FF             ; loads ACCb = 7FFF
0044 0032                       movwf   ACCbLO
                        ;
0045 0907                       call    D_mpyS          ; Here (ACCb,ACCc) = 00FF 7E01
                        ;
0046 0A46               self    goto    self
                        ;
                                org     PIC54
01FF 0A3D                       goto    main
                                END
                        ;*****************************************************************

Errors   :     0
Warnings :     0
```

5

© 1994 Microchip Technology Inc.

APPENDIX D: DOUBLE PRECISION MULTIPLICATION LISTINGS (FAST)

```
MPASM B0.54                                                    PAGE  1

                ;*****************************************************************
                ;                       Double Precision Multiplication
                ;
                ;               ( Optimized for Speed : straight Line Code )
                ;
                ;*****************************************************************;
                ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
                ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                ;       (c) CALL D_mpy
                ;       (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
                ;
                ;   Performance :
                ;               Program Memory  :       240
                ;               Clock Cycles    :       233
                ;
                ;       Note : The above timing is the worst case timing, when the
                ;               register ACCb = FFFF. The speed may be improved if
                ;               the register ACCb contains a number ( out of the two
                ;               numbers ) with less number of 1s.
                ;
                ;               The performance specs are for Unsigned arithmetic ( i.e,
                ;               with "SIGNED equ  FALSE ").

                ;*****************************************************************;
                ;
                        LIST    p=16c54
0010            ACCaLO  equ     10
0011            ACCaHI  equ     11
0012            ACCbLO  equ     12
0013            ACCbHI  equ     13
0014            ACCcLO  equ     14
0015            ACCcHI  equ     15
0016            ACCdLO  equ     16
0017            ACCdHI  equ     17
0018            temp    equ     18
0019            sign    equ     19
                ;
                  include "mpreg.h"
                ;*************************       PIC16C5X Header ************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
```


© 1994 Microchip Technology Inc.

```
0003                    PD      equ     3h
0004                    T_OUT   equ     4h
0004                    TO      equ     4h
0005                    PA0     equ     5h
0006                    PA1     equ     6h
0007                    PA2     equ     7h
                        ;
0001                    Same    equ     1h
                        ;
0000                    LSB     equ     0h
0007                    MSB     equ     7h
                        ;
0001                    TRUE    equ     1h
0001                    YES     equ     1h
0000                    FALSE   equ     0h
0000                    NO      equ     0h
                        ;
                        ;*****************************************************************************


                              org     0
                        ;*****************************************************************
0000                    SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
                        ;                               ; for Multiplication & Division needs
                        ;                               ; to be assembled as Signed Integer
                        ;                               ; Routines. If 'FALSE' the above two
                        ;                               ; routines ( D_mpy & D_div ) use
                        ;                               ; unsigned arithmetic.
                        ;*****************************************************************
                        ;       multiplication macro
                        ;
                        mulMac  MACRO
                          LOCAL   NO_ADD
                        ;
                          rrf     ACCdHI                ;rotate d right
                          rrf     ACCdLO
                          btfss   STATUS,CARRY          ;need to add?
                          goto    NO_ADD                ; no addition necessary
                          movf    ACCaLO,w              ; Addition ( ACCb + ACCa -> ACCb )
                          addwf   ACCbLO                ;add lsb
                          btfsc   STATUS,CARRY          ;add in carry
                          incf    ACCbHI
                          movf    ACCaHI,w
                          addwf   ACCbHI                ;add msb
                        NO_ADD  rrf     ACCbHI
                          rrf     ACCbLO
                          rrf     ACCcHI
                          rrf     ACCcLO
                        ;
                          ENDM
                        ;
                        ;*****************************************************************;
                        ;               Double Precision Multiply ( 16x16 -> 32 )
                        ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                        ;  in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                        ;
                        D_mpyF                          ;results in ACCb(16 msb's) and ACCc(16 lsb's)
                        ;
                            IF   SIGNED
                            CALL    S_SIGN
                            ENDIF
                        ;
0000 09E2                       call    setup
                        ;
```

```
                        ; use the mulMac macro 16 times
                        ;
                          mulMac
                               LOCAL   NO_ADD
                        ;
0001 0337                       rrf     ACCdHI          ;rotate d right
0002 0336                       rrf     ACCdLO
0003 0703                       btfss   STATUS,CARRY    ;need to add?
0004 0A0B                       goto    NO_ADD          ; no addition necessary
0005 0210                       movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0006 01F2                       addwf   ACCbLO          ;add lsb
0007 0603                       btfsc   STATUS,CARRY    ;add in carry
0008 02B3                       incf    ACCbHI
0009 0211                       movf    ACCaHI,w
000A 01F3                       addwf   ACCbHI            ;add msb
000B 0333               NO_ADD  rrf     ACCbHI
000C 0332                       rrf     ACCbLO
000D 0335                       rrf     ACCcHI
000E 0334                       rrf     ACCcLO
                        ;

                          mulMac
                               LOCAL   NO_ADD
                        ;
000F 0337                       rrf     ACCdHI          ;rotate d right
0010 0336                       rrf     ACCdLO
0011 0703                       btfss   STATUS,CARRY    ;need to add?
0012 0A19                       goto    NO_ADD          ; no addition necessary
0013 0210                       movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0014 01F2                       addwf   ACCbLO          ;add lsb
0015 0603                       btfsc   STATUS,CARRY    ;add in carry
0016 02B3                       incf    ACCbHI
0017 0211                       movf    ACCaHI,w
0018 01F3                       addwf   ACCbHI          ;add msb
0019 0333               NO_ADD  rrf     ACCbHI
001A 0332                       rrf     ACCbLO
001B 0335                       rrf     ACCcHI
001C 0334                       rrf     ACCcLO
                        ;

                          mulMac
                               LOCAL   NO_ADD
                        ;
001D 0337                       rrf     ACCdHI          ;rotate d right
001E 0336                       rrf     ACCdLO
001F 0703                       btfss   STATUS,CARRY    ;need to add?
0020 0A27                       goto    NO_ADD          ; no addition necessary
0021 0210                       movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0022 01F2                       addwf   ACCbLO          ;add lsb
0023 0603                       btfsc   STATUS,CARRY    ;add in carry
0024 02B3                       incf    ACCbHI
0025 0211                       movf    ACCaHI,w
0026 01F3                       addwf   ACCbHI          ;add msb
0027 0333               NO_ADD  rrf     ACCbHI
0028 0332                       rrf     ACCbLO
0029 0335                       rrf     ACCcHI
002A 0334                       rrf     ACCcLO
                        ;
```


© 1994 Microchip Technology Inc.

```
                        mulMac
                             LOCAL   NO_ADD
                        ;
002B 0337                    rrf     ACCdHI          ;rotate d right
002C 0336                    rrf     ACCdLO
002D 0703                    btfss   STATUS,CARRY    ;need to add?
002E 0A35                    goto    NO_ADD          ; no addition necessary
002F 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0030 01F2                    addwf   ACCbLO          ;add lsb
0031 0603                    btfsc   STATUS,CARRY    ;add in carry
0032 02B3                    incf    ACCbHI
0033 0211                    movf    ACCaHI,w
0034 01F3                    addwf   ACCbHI          ;add msb
0035 0333               NO_ADD  rrf     ACCbHI
0036 0332                    rrf     ACCbLO
0037 0335                    rrf     ACCcHI
0038 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
0039 0337                    rrf     ACCdHI          ;rotate d right
003A 0336                    rrf     ACCdLO
003B 0703                    btfss   STATUS,CARRY    ;need to add?
003C 0A43                    goto    NO_ADD          ; no addition necessary
003D 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
003E 01F2                    addwf   ACCbLO          ;add lsb
003F 0603                    btfsc   STATUS,CARRY    ;add in carry
0040 02B3                    incf    ACCbHI
0041 0211                    movf    ACCaHI,w
0042 01F3                    addwf   ACCbHI          ;add msb
0043 0333               NO_ADD  rrf     ACCbHI
0044 0332                    rrf     ACCbLO
0045 0335                    rrf     ACCcHI
0046 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
0047 0337                    rrf     ACCdHI          ;rotate d right
0048 0336                    rrf     ACCdLO
0049 0703                    btfss   STATUS,CARRY    ;need to add?
004A 0A51                    goto    NO_ADD          ; no addition necessary
004B 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
004C 01F2                    addwf   ACCbLO          ;add lsb
004D 0603                    btfsc   STATUS,CARRY    ;add in carry
004E 02B3                    incf    ACCbHI
004F 0211                    movf    ACCaHI,w
0050 01F3                    addwf   ACCbHI          ;add msb
0051 0333               NO_ADD  rrf     ACCbHI
0052 0332                    rrf     ACCbLO
0053 0335                    rrf     ACCcHI
0054 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
0055 0337                    rrf     ACCdHI          ;rotate d right
0056 0336                    rrf     ACCdLO
0057 0703                    btfss   STATUS,CARRY    ;need to add?
0058 0A5F                    goto    NO_ADD          ; no addition necessary
0059 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
005A 01F2                    addwf   ACCbLO          ;add lsb
005B 0603                    btfsc   STATUS,CARRY    ;add in carry
005C 02B3                    incf    ACCbHI
005D 0211                    movf    ACCaHI,w
005E 01F3                    addwf   ACCbHI            ;add msb
```

5

© 1994 Microchip Technology Inc.

```
005F 0333           NO_ADD  rrf     ACCbHI
0060 0332                   rrf     ACCbLO
0061 0335                   rrf     ACCcHI
0062 0334                   rrf     ACCcLO
                    ;
                      mulMac
                              LOCAL   NO_ADD
                    ;
0063 0337                   rrf     ACCdHI          ;rotate d right
0064 0336                   rrf     ACCdLO
0065 0703                   btfss   STATUS,CARRY    ;need to add?
0066 0A6D                   goto    NO_ADD          ; no addition necessary
0067 0210                   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0068 01F2                   addwf   ACCbLO          ;add lsb
0069 0603                   btfsc   STATUS,CARRY    ;add in carry
006A 02B3                   incf    ACCbHI
006B 0211                   movf    ACCaHI,w
006C 01F3                   addwf   ACCbHI          ;add msb
006D 0333           NO_ADD  rrf     ACCbHI
006E 0332                   rrf     ACCbLO
006F 0335                   rrf     ACCcHI
0070 0334                   rrf     ACCcLO
                    ;
                      mulMac
                              LOCAL   NO_ADD
                    ;
0071 0337                   rrf     ACCdHI          ;rotate d right
0072 0336                   rrf     ACCdLO
0073 0703                   btfss   STATUS,CARRY    ;need to add?
0074 0A7B                   goto    NO_ADD          ; no addition necessary
0075 0210                   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0076 01F2                   addwf   ACCbLO          ;add lsb
0077 0603                   btfsc   STATUS,CARRY    ;add in carry
0078 02B3                   incf    ACCbHI
0079 0211                   movf    ACCaHI,w
007A 01F3                   addwf   ACCbHI          ;add msb
007B 0333           NO_ADD  rrf     ACCbHI
007C 0332                   rrf     ACCbLO
007D 0335                   rrf     ACCcHI
007E 0334                   rrf     ACCcLO
                    ;
                      mulMac
                              LOCAL   NO_ADD
                    ;
007F 0337                   rrf     ACCdHI          ;rotate d right
0080 0336                   rrf     ACCdLO
0081 0703                   btfss   STATUS,CARRY    ;need to add?
0082 0A89                   goto    NO_ADD          ; no addition necessary
0083 0210                   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0084 01F2                   addwf   ACCbLO          ;add lsb
0085 0603                   btfsc   STATUS,CARRY    ;add in carry
0086 02B3                   incf    ACCbHI
0087 0211                   movf    ACCaHI,w
0088 01F3                   addwf   ACCbHI          ;add msb
0089 0333           NO_ADD  rrf     ACCbHI
008A 0332                   rrf     ACCbLO
008B 0335                   rrf     ACCcHI
008C 0334                   rrf     ACCcLO
                    ;
```


© 1994 Microchip Technology Inc.

```
                        mulMac
                             LOCAL   NO_ADD
                        ;
008D 0337                    rrf     ACCdHI          ;rotate d right
008E 0336                    rrf     ACCdLO
008F 0703                    btfss   STATUS,CARRY    ;need to add?
0090 0A97                    goto    NO_ADD          ; no addition necessary
0091 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
0092 01F2                    addwf   ACCbLO          ;add lsb
0093 0603                    btfsc   STATUS,CARRY    ;add in carry
0094 02B3                    incf    ACCbHI
0095 0211                    movf    ACCaHI,w
0096 01F3                    addwf   ACCbHI          ;add msb
0097 0333               NO_ADD  rrf     ACCbHI
0098 0332                    rrf     ACCbLO
0099 0335                    rrf     ACCcHI
009A 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
009B 0337                    rrf     ACCdHI          ;rotate d right
009C 0336                    rrf     ACCdLO
009D 0703                    btfss   STATUS,CARRY    ;need to add?
009E 0AA5                    goto    NO_ADD          ; no addition necessary
009F 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
00A0 01F2                    addwf   ACCbLO          ;add lsb
00A1 0603                    btfsc   STATUS,CARRY    ;add in carry
00A2 02B3                    incf    ACCbHI
00A3 0211                    movf    ACCaHI,w
00A4 01F3                    addwf   ACCbHI          ;add msb
00A5 0333               NO_ADD  rrf     ACCbHI
00A6 0332                    rrf     ACCbLO
00A7 0335                    rrf     ACCcHI
00A8 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
00A9 0337                    rrf     ACCdHI          ;rotate d right
00AA 0336                    rrf     ACCdLO
00AB 0703                    btfss   STATUS,CARRY    ;need to add?
00AC 0AB3                    goto    NO_ADD          ; no addition necessary
00AD 0210                    movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
00AE 01F2                    addwf   ACCbLO          ;add lsb
00AF 0603                    btfsc   STATUS,CARRY    ;add in carry
00B0 02B3                    incf    ACCbHI
00B1 0211                    movf    ACCaHI,w
00B2 01F3                    addwf   ACCbHI          ;add msb
00B3 0333               NO_ADD  rrf     ACCbHI
00B4 0332                    rrf     ACCbLO
00B5 0335                    rrf     ACCcHI
00B6 0334                    rrf     ACCcLO
                        ;
                        mulMac
                             LOCAL   NO_ADD
                        ;
00B7 0337                    rrf     ACCdHI          ;rotate d right
00B8 0336                    rrf     ACCdLO
00B9 0703                    btfss   STATUS,CARRY    ;need to add?
00BA 0AC1                    goto    NO_ADD          ;no addition necessary
00BB 0210                    movf    ACCaLO,w        ;Addition ( ACCb + ACCa -> ACCb )
00BC 01F2                    addwf   ACCbLO          ;add lsb
00BD 0603                    btfsc   STATUS,CARRY    ;add in carry
00BE 02B3                    incf    ACCbHI
00BF 0211                    movf    ACCaHI,w
00C0 01F3                    addwf   ACCbHI          ;add msb
```

5

© 1994 Microchip Technology Inc.

```
00C1 0333           NO_ADD  rrf     ACCbHI
00C2 0332                   rrf     ACCbLO
00C3 0335                   rrf     ACCcHI
00C4 0334                   rrf     ACCcLO
                    ;
                      mulMac
                            LOCAL   NO_ADD
                    ;
00C5 0337                   rrf     ACCdHI          ;rotate d right
00C6 0336                   rrf     ACCdLO
00C7 0703                   btfss   STATUS,CARRY    ;need to add?
00C8 0ACF                   goto    NO_ADD          ; no addition necessary
00C9 0210                   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
00CA 01F2                   addwf   ACCbLO          ;add lsb
00CB 0603                   btfsc   STATUS,CARRY    ;add in carry
00CC 02B3                   incf    ACCbHI
00CD 0211                   movf    ACCaHI,w
00CE 01F3                   addwf   ACCbHI          ;add msb
00CF 0333           NO_ADD  rrf     ACCbHI
00D0 0332                   rrf     ACCbLO
00D1 0335                   rrf     ACCcHI
00D2 0334                   rrf     ACCcLO
                    ;
                      mulMac
                            LOCAL   NO_ADD
                    ;
00D3 0337                   rrf     ACCdHI          ;rotate d right
00D4 0336                   rrf     ACCdLO
00D5 0703                   btfss   STATUS,CARRY    ;need to add?
00D6 0ADD                   goto    NO_ADD          ; no addition necessary
00D7 0210                   movf    ACCaLO,w        ; Addition ( ACCb + ACCa -> ACCb )
00D8 01F2                   addwf   ACCbLO          ;add lsb
00D9 0603                   btfsc   STATUS,CARRY    ;add in carry
00DA 02B3                   incf    ACCbHI
00DB 0211                   movf    ACCaHI,w
00DC 01F3                   addwf   ACCbHI          ;add msb
00DD 0333           NO_ADD  rrf     ACCbHI
00DE 0332                   rrf     ACCbLO
00DF 0335                   rrf     ACCcHI
00E0 0334                   rrf     ACCcLO
                    ;
                    ;
                       IF     SIGNED
                            btfss   sign,MSB
                            retlw   0
                            comf    ACCcLO          ; negate ACCa ( -ACCa -> ACCa )
                            incf    ACCcLO
                            btfsc   STATUS,Z_bit
                            decf    ACCcHI
                            comf    ACCcHI
                            btfsc   STATUS,Z_bit
                    neg_B   comf    ACCbLO          ; negate ACCb
                            incf    ACCbLO
                            btfsc   STATUS,Z_bit
                            decf    ACCbHI
                            comf    ACCbHI
                            retlw   0
                         ELSE
00E1 0800                   retlw   0
                         ENDIF
```


© 1994 Microchip Technology Inc.

```
                        ;
                        ;*****************************************************************
                        ;
00E2 0C10               setup   movlw   .16             ; for 16 shifts
00E3 0038                       movwf   temp
00E4 0213                       movf    ACCbHI,w        ;move ACCb to ACCd
00E5 0037                       movwf   ACCdHI
00E6 0212                       movf    ACCbLO,w
00E7 0036                       movwf   ACCdLO
00E8 0073                       clrf    ACCbHI
00E9 0072                       clrf    ACCbLO
00EA 0800                       retlw   0
                        ;
                        ;*****************************************************************
                        ;
00EB 0270               neg_A   comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
00EC 02B0                       incf    ACCaLO
00ED 0643                       btfsc   STATUS,Z_bit
00EE 00F1                       decf    ACCaHI
00EF 0271                       comf    ACCaHI
00F0 0800                       retlw   0
                        ;
                        ;*****************************************************************
                        ;  Assemble this section only if Signed Arithmetic Needed
                        ;
                            IF    SIGNED
                        ;
                        S_SIGN  movf    ACCaHI,W
                                xorwf   ACCbHI,W
                                movwf   sign
                                btfss   ACCbHI,MSB      ; if MSB set go & negate ACCb
                                goto    chek_A
                        ;
                                comf    ACCbLO          ; negate ACCb
                                incf    ACCbLO
                                btfsc   STATUS,Z_bit
                                decf    ACCbHI
                                comf    ACCbHI
                        ;
                        chek_A  btfss   ACCaHI,MSB      ; if MSB set go & negate ACCa
                                retlw   0
                                goto    neg_A
                        ;
                            ENDIF
                        ;
```


© 1994 Microchip Technology Inc.

```
                ;*****************************************************************
                ;                          Test Program
                ;*****************************************************************
                ;    Load constant values to ACCa & ACCb for testing
                ;
00F1 0C01       loadAB  movlw   1
00F2 0031               movwf   ACCaHI
00F3 0CFF               movlw   0FF             ; loads ACCa = 01FF
00F4 0030               movwf   ACCaLO
                ;
00F5 0C7F               movlw   07F
00F6 0033               movwf   ACCbHI
00F7 0CFF               movlw   0FF             ; loads ACCb = 7FFF
00F8 0032               movwf   ACCbLO
00F9 0800               retlw   0
                ;
00FA 0000       main    nop
                ;
00FB 09F1               call    loadAB          ; result of multiplying ACCb*ACCa-
00FC 0900               call    D_mpyF          ; Here (ACCb,ACCc) = 00FF 7E01
                ;
00FD 0AFD       self    goto    self
                ;
                        org     PIC54
01FF 0AFA               goto    main
                        END
                ;*****************************************************************


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX E: DOUBLE PRECISION ADDITION AND SUBTRACTION LISTING

```
MPASM B0.54                                                    PAGE  1

                ;*****************************************************************
                ;                Double Precision Addition & Subtraction
                ;
                ;*****************************************************************;
                ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                ;      (c) CALL D_add
                ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                ;
                ;   Performance :
                ;               Program Memory  :       07
                ;               Clock Cycles    :       08
                ;*****************************************************************;
                ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                ;      (c) CALL D_sub
                ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                ;
                ;   Performance :
                ;               Program Memory  :       14
                ;               Clock Cycles    :       17
                ;*****************************************************************;
                ;
                      LIST  P=16C54
0010            ACCaLO  equ     10
0011            ACCaHI  equ     11
0012            ACCbLO  equ     12
0013            ACCbHI  equ     13
                ;
                  include "mpreg.h"
                ;*************************       PIC16C5X Header *************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
                ;
```

```
0001                Same    equ     1h
                    ;
0000                LSB     equ     0h
0007                MSB     equ     7h
                    ;
0001                TRUE    equ     1h
0001                YES     equ     1h
0000                FALSE   equ     0h
0000                NO      equ     0h
                    ;
                    ;*****************************************************************************

                            org     0
                    ;*****************************************************************
                    ;         Double Precision Subtraction ( ACCb - ACCa -> ACCb )
                    ;
0000 0908           D_sub   call    neg_A           ; At first negate ACCa; Then add
                    ;
                    ;*****************************************************************
                    ;  Double Precision  Addition ( ACCb + ACCa -> ACCb )
                    ;
0001 0210           D_add   movf    ACCaLO,w
0002 01F2                   addwf   ACCbLO          ;add lsb
0003 0603                   btfsc   STATUS,CARRY    ;add in carry
0004 02B3                   incf    ACCbHI
0005 0211                   movf    ACCaHI,w
0006 01F3                   addwf   ACCbHI          ;add msb
0007 0800                   retlw   0
                    ;
                    ;
0008 0270           neg_A   comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
0009 02B0                   incf    ACCaLO
000A 0643                   btfsc   STATUS,Z_bit
000B 00F1                   decf    ACCaHI
000C 0271                   comf    ACCaHI
000D 0800                   retlw   0
                    ;
                    ;*****************************************************************
                    ;                       Test Program
                    ;*****************************************************************
                    ;    Load constant values to ACCa & ACCb for testing
                    ;
000E 0C01           loadAB  movlw   1
000F 0031                   movwf   ACCaHI
0010 0CFF                   movlw   0FF             ; loads ACCa = 01FF
0011 0030                   movwf   ACCaLO
                    ;
0012 0C7F                   movlw   07F
0013 0033                   movwf   ACCbHI
0014 0CFF                   movlw   0FF             ; loads ACCb = 7FFF
0015 0032                   movwf   ACCbLO
0016 0800                   retlw   0
                    ;
0017 0000           main    nop
                    ;
0018 090E                   call    loadAB          ; result of adding ACCb+ACCa->ACCb
0019 0901                   call    D_add           ; Here Accb = 81FE
                    ;
001A 090E                   call    loadAB          ; result of subtracting ACCb - ACCa->ACCb
001B 0900                   call    D_sub           ; Here Accb = 7E00
                    ;
001C 0A1C           self    goto    self
                    ;
                            org     PIC54
01FF 0A17                   goto    main
                            END
                    ;*****************************************************************
Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX F: BCD TO BINARY CONVERSION LISTING

```
MPASM B0.54                                                    PAGE  1

                    ;
                    ;*****************************************************************
                    ;               BCD To Binary Conversion
                    ;
                    ;       This routine converts a 5 digit BCD number to a 16 bit binary
                    ; number.
                    ;       The input 5 digit BCD numbers are asumed to be in locations
                    ; R0, R1 & R2 with R0 containing the MSD in its right most nibble.
                    ;
                    ;       The 16 bit binary number is output in registers H_byte & L_byte
                    ; ( high byte & low byte repectively ).
                    ;
                    ;       The method used for conversion is :
                    ;               input number X = abcde ( the 5 digit BCD number )
                    ;               X = abcde = 10[10[10[10a+b]+c]+d]+e
                    ;
                    ;   Performance :
                    ;               Program Memory  :       30
                    ;               Clock Cycles    :       121
                    ;
                    ;*****************************************************************;
                    ;
                          LIST    P=16C54
0010                H_byte  equ     10
0011                L_byte  equ     11
0012                R0      equ     12              ; RAM Assignments
0013                R1      equ     13
0014                R2      equ     14
                    ;
0015                H_temp  equ     15              ; temporary register
0016                L_temp  equ     16              ; temporary register
                    ;
                    ;
                       INCLUDE         "mpreg.h"
                    ;*************************       PIC16C5X Header ************************
01FF                PIC54   equ     1FFH            ; Define Reset Vectors
01FF                PIC55   equ     1FFH
03FF                PIC56   equ     3FFH
07FF                PIC57   equ     7FFH
                    ;
0001                RTCC    equ     1h
0002                PC      equ     2h
0003                STATUS  equ     3h              ; F3 Reg is STATUS Reg.
0004                FSR     equ     4h
                    ;
0005                Port_A  equ     5h
0006                Port_B  equ     6h              ; I/O Port Assignments
0007                Port_C  equ     7h
                    ;
                    ;*****************************************************************
                    ;
                    ;                               ; STATUS REG. Bits
0000                CARRY   equ     0h              ; Carry Bit is Bit.0 of F3
0000                C       equ     0h
0001                DCARRY  equ     1h
0001                DC      equ     1h
0002                Z_bit   equ     2h              ; Bit 2 of F3 is Zero Bit
0002                Z       equ     2h
0003                P_DOWN  equ     3h
0003                PD      equ     3h
0004                T_OUT   equ     4h
0004                TO      equ     4h
0005                PA0     equ     5h
0006                PA1     equ     6h
```

```
0007                    PA2     equ     7h
                        ;
0001                    Same    equ     1h
                        ;
0000                    LSB     equ     0h
0007                    MSB     equ     7h
                        ;
0001                    TRUE    equ     1h
0001                    YES     equ     1h
0000                    FALSE   equ     0h
0000                    NO      equ     0h
                        ;
                        ;*****************************************************************************
                        ;
                        ;
0000 0E0F               mpy10b  andlw   0F
0001 01F1                       addwf   L_byte
0002 0603                       btfsc   STATUS,CARRY
0003 02B0                       incf    H_byte
0004 0403               mpy10a  bcf     STATUS,CARRY    ; multiply by 2
0005 0351                       rlf     L_byte,w
0006 0036                       movwf   L_temp
0007 0350                       rlf     H_byte,w        ; (H_temp,L_temp) = 2*N
0008 0035                       movwf   H_temp
                        ;
0009 0403                       bcf     STATUS,CARRY    ; multiply by 2
000A 0371                       rlf     L_byte
000B 0370                       rlf     H_byte
000C 0403                       bcf     STATUS,CARRY    ; multiply by 2
000D 0371                       rlf     L_byte
000E 0370                       rlf     H_byte
000F 0403                       bcf     STATUS,CARRY    ; multiply by 2
0010 0371                       rlf     L_byte
0011 0370                       rlf     H_byte          ; (H_byte,L_byte) = 8*N
                        ;
0012 0216                       movf    L_temp,w
0013 01F1                       addwf   L_byte
0014 0603                       btfsc   STATUS,CARRY
0015 02B0                       incf    H_byte
0016 0215                       movf    H_temp,w
0017 01F0                       addwf   H_byte
0018 0800                       retlw   0               ; (H_byte,L_byte) = 10*N
                        ;
                        ;
0019 0070               BCDtoB  clrf    H_byte
001A 0212                       movf    R0,w
001B 0E0F                       andlw   0F
001C 0031                       movwf   L_byte
001D 0904                       call    mpy10a          ; result = 10a+b
                        ;
001E 0393                       swapf   R1,w
001F 0900                       call    mpy10b          ; result = 10[10a+b]
                        ;
0020 0213                       movf    R1,w
0021 0900                       call    mpy10b          ; result = 10[10[10a+b]+c]
                        ;
0022 0394                       swapf   R2,w
0023 0900                       call    mpy10b          ; result = 10[10[10[10a+b]+c]+d]
                        ;
0024 0214                       movf    R2,w
0025 0E0F                       andlw   0F
0026 01F1                       addwf   L_byte
0027 0603                       btfsc   STATUS,CARRY
0028 02B0                       incf    H_byte          ; result = 10[10[10[10a+b]+c]+d]+e
0029 0800                       retlw   0               ; BCD to binary conversion done
                        ;
                        ;
```


© 1994 Microchip Technology Inc.

```
                ;*****************************************************************
                ;                Test Program
                ;*****************************************************************
002A 0C06          main    movlw   06
002B 0032                  movwf   R0      ; Set R0 = 06
002C 0C55                  movlw   55
002D 0033                  movwf   R1      ; Set R1 = 55
002E 0C35                  movlw   35
002F 0034                  movwf   R2      ; Set R2 = 35      ( R0, R1, R2 = 6,55,35 )
                ;
0030 0919                  call    BCDtoB  ; After conversion H_Byte = FF & L_Byte = FF
                ;
0031 0A31          self    goto    self
                ;
                           org     1FF
01FF 0A2A                  goto    main
                ;
                           END


Errors   :     0
Warnings :     0
```

5

APPENDIX G: BINARY (8-BIT) TO BCD LISTING

```
MPASM B0.54                                                    PAGE  1

                        LIST    p=16c54,n=0
                        ;
                        ;*****************************************************************
                        ;                   Binary To BCD Conversion Routine
                        ;
                        ;       This routine converts the 8 bit binary number in the W Register
                        ; to a 2 digit BCD number.
                        ;       The least significant digit is returned in location LSD and
                        ; the most significant digit is returned in location MSD.
                        ;
                        ;   Performance :
                        ;                Program Memory  :       10
                        ;                Clock Cycles    :       81 (worst case when W = 63 Hex )
                        ;                                           ( i.e max Decimal number 99 )
                        ;*****************************************************************
                        ;
0010                    LSD     equ     10
0011                    MSD     equ     11
                        ;
        INCLUDE           "mpreg.h"
                        ;************************  PIC16C5X Header    ************************
01FF                    PIC54   equ     1FFH    ; Define Reset Vectors
01FF                    PIC55   equ     1FFH
03FF                    PIC56   equ     3FFH
07FF                    PIC57   equ     7FFH
                        ;
0001                    RTCC    equ     1
0002                    PC      equ     2
0003                    STATUS  equ     3       ; F3 Reg is STATUS Reg.
0004                    FSR     equ     4
                        ;
0005                    Port_A  equ     5
0006                    Port_B  equ     6       ; I/O Port Assignments
0007                    Port_C  equ     7
                        ;
                        ;*******************************************************************************
                        ;
                                                ; STATUS REG. Bits
0000                    CARRY   equ     0       ; Carry Bit is Bit.0 of F3
0000                    C       equ     0
0001                    DCARRY  equ     1
0001                    DC      equ     1
0002                    Z_bit   equ     2       ; Bit 2 of F3 is Zero Bit
0002                    Z       equ     2
0003                    P_DOWN  equ     3
0003                    PD      equ     3
0004                    T_OUT   equ     4
0004                    TO      equ     4
0005                    PA0     equ     5
0006                    PA1     equ     6
0007                    PA2     equ     7
                        ;
0001                    Same    equ     1
0000                    W       equ     0
                        ;
0000                    LSB     equ     0
0007                    MSB     equ     7
                        ;
0001                    TRUE    equ     1
```


© 1994 Microchip Technology Inc.

```
0001                    YES     equ     1
0000                    FALSE   equ     0
0000                    NO      equ     0
                        ;
                        ;*****************************************************************************


                        ;
0000 0071               BinBCD  clrf    MSD
0001 0030                       movwf   LSD
0002 0C0A               gtenth  movlw   .10
0003 0090                       subwf   LSD,W
0004 0703                       BTFSS   STATUS,CARRY
0005 0A09                       goto    over
0006 0030                       movwf   LSD
0007 02B1                       incf    MSD
0008 0A02                       goto    gtenth
0009 0800               over    retlw   0
                        ;*****************************************************************
                        ;
000A 0C63               main    movlw   63              ; W reg = 63 Hex
000B 0900                       call    BinBCD          ; after conversion, MSD = 9 & LSD = 9
000C 0A0C               self    goto    self            ; ( 63 Hex = 99 Decimal )
                        ;
                                org     1FF
01FF 0A0A                       goto    main
                        ;
                                END

Errors   :     0
Warnings :     0
```

5

APPENDIX H: BINARY (16-BIT) TO BCD LISTING

```
MPASM B0.54                                                    PAGE  1

                ;
                ;*****************************************************************
                ;                   Binary To BCD Conversion Routine
                ;      This routine converts a 16 Bit binary Number to a 5 Digit
                ; BCD Number. This routine is useful since PIC16C55 & PIC16C57
                ; have  two 8 bit ports and one 4 bit port ( total of 5 BCD digits)
                ;
                ;       The 16 bit binary number is input in locations H_byte and
                ; L_byte with the high byte in H_byte.
                ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
                ; containing the MSD in its right most nibble.
                ;
                ;   Performance :
                ;               Program Memory  :       35
                ;               Clock Cycles    :       885
                ;
                ;*****************************************************************;
                ;
                        LIST    P=16C54
0016            count   equ     16
0017            temp    equ     17
                ;
0010            H_byte  equ     10
0011            L_byte  equ     11
0012            R0      equ     12              ; RAM Assignments
0013            R1      equ     13
0014            R2      equ     14
                ;
                  include         "mpreg.h"
                ;*************************       PIC16C5X Header ************************
01FF            PIC54   equ     1FFH            ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h              ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h              ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************************
                ;
                ;                               ; STATUS REG. Bits
0000            CARRY   equ     0h              ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h              ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
                ;
0001            Same    equ     1h
                ;
0000            LSB     equ     0h
```


© 1994 Microchip Technology Inc.

```
0007                MSB     equ     7h
                    ;
0001                TRUE    equ     1h
0001                YES     equ     1h
0000                FALSE   equ     0h
0000                NO      equ     0h
                    ;
                    ;*****************************************************************
                    ;
0000 0403           B2_BCD  bcf     STATUS,0                ; clear the carry bit
0001 0C10                   movlw   .16
0002 0036                   movwf   count
0003 0072                   clrf    R0
0004 0073                   clrf    R1
0005 0074                   clrf    R2
0006 0371           loop16  rlf     L_byte
0007 0370                   rlf     H_byte
0008 0374                   rlf     R2
0009 0373                   rlf     R1
000A 0372                   rlf     R0
                    ;
000B 02F6                   decfsz  count
000C 0A0E                   goto    adjDEC
000D 0800                   RETLW   0
                    ;
000E 0C14           adjDEC  movlw   R2
000F 0024                   movwf   FSR
0010 0918                   call    adjBCD
                    ;
0011 0C13                   movlw   R1
0012 0024                   movwf   FSR
0013 0918                   call    adjBCD
                    ;
0014 0C12                   movlw   R0
0015 0024                   movwf   FSR
0016 0918                   call    adjBCD
                    ;
0017 0A06                   goto    loop16
                    ;
0018 0C03           adjBCD  movlw   3
0019 01C0                   addwf   0,W
001A 0037                   movwf   temp
001B 0677                   btfsc   temp,3          ; test if result > 7
001C 0020                   movwf   0
001D 0C30                   movlw   30
001E 01C0                   addwf   0,W
001F 0037                   movwf   temp
0020 06F7                   btfsc   temp,7          ; test if result > 7
0021 0020                   movwf   0               ; save as MSD
0022 0800                   RETLW   0
                    ;
                    ;*****************************************************************
                    ;               Test Program
                    ;*****************************************************************
0023 0CFF           main    movlw   0FF
0024 0030                   movwf   H_byte
0025 0031                   movwf   L_byte          ; The 16 bit binary number = FFFF
0026 0900                   call    B2_BCD          ; After conversion the Decimal Number
                    ;                               ; in R0,R1,R2 = 06,55,35
                    ;
0027 0A27           self    goto    self
                    ;
                          org     1FF
01FF 0A23                   goto    main
                    ;
                          END
```

APPENDIX I: UNSIGNED BCD ADDITION LISTING

```
MPASM B0.54                                                    PAGE  1

                ;******************  Unsigned BCD Addition   ***************
                ;
                ;       This routine performs a 2 Digit Unsigned BCD Addition
                ; It is assumed that the two BCD numbers to be added are in
                ; locations Num_1 & Num_2. The result is the sum of Num_1+Num_2
                ; and is stored in location Num_2 and the overflow carry is returned
                ; in location Num_1
                ;
                ;   Performance :
                ;               Program Memory  :       25
                ;               Clock Cycles    :       23    ( worst case )
                ;
                ;       Rev 2.0 changed on 7/30/92.
                ;
                ;*****************************************************************;
                ;
                        LIST    P=16C54
0008            Num_1   equ     8       ; Overflow flow carry overwrites Num_1
0008            result  equ     8
                ;
0009            Num_2   equ     9       ; Num_2 + Num_1 overwrites Num_2
0009            O_flow  equ     9
                ;
                ;
                  include         "mpreg.h"
                ;*************************       PIC16C5X Header *************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
                ;
0001             Same    equ     1h
                ;
0000            LSB     equ     0h
0007            MSB     equ     7h
                ;
0001            TRUE    equ     1h
0001            YES     equ     1h
0000            FALSE   equ     0h
```


© 1994 Microchip Technology Inc.

```
0000                    NO      equ     0h
                        ;
                        ;*****************************************************************************

                        ;
0000 0208               BCDAdd  movf    Num_1,w
0001 01E9                       addwf   Num_2           ; do binary addition
0002 0068                       clrf    Num_1
0003 0368                       rlf     Num_1
0004 0623                       btfsc   STATUS,DC       ; Is DC = 0 ?
0005 0A0D                       goto    adjust          ; adjust LSD
0006 0C06                       movlw   6
0007 01E9                       addwf   Num_2           ; Test for LSD > 9 ( by adding 6
0008 0603                       btfsc   STATUS,CARRY
0009 02A8                       incf    Num_1
000A 0723                       btfss   STATUS,DC       ; & checking Digit Carry
000B 00A9                       subwf   Num_2           ; LSD < 9 , so get back original value.
000C 0A0F                       goto    over1
000D 0C06               adjust  movlw   6
000E 01E9                        addwf   Num_2
000F 0C60               over1   movlw   60              ; add 6 to MSD
0010 01E9                       addwf   Num_2
0011 0603                       btfsc   STATUS,CARRY
0012 0A16                       goto    over3
0013 0708                       btfss   Num_1,0
0014 00A9                       subwf   Num_2
0015 0800                       RETLW   0
0016 0C01               over3   movlw   1
0017 0028                       movwf   Num_1
0018 0800                       RETLW   0
                        ;
                        ;*****************************************************************
                        ;               Test Program
                        ;*****************************************************************
0019 0C99               main    movlw   99
001A 0028                       movwf   Num_1           ; Set Num_1 = 99 ( max BCD digit )
001B 0C99                       movlw   99
001C 0029                       movwf   Num_2           ; Set Num_2 = 99
                        ;
001D 0900                       call    BCDAdd          ; After addition, Num_2 = 98
                        ;                               ;  and Num_1 = 01 ( 99+99 = 198 -> max number
)
                        ;
001E 0A1E               self    goto    self
                        ;
                              org     1FF
01FF 0A19                      goto    main
                        ;
                              END



Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX J: UNSIGNED BCD SUBTRACTION LISTING

```
MPASM B0.54                                                    PAGE  1

                ;******************  Unsigned BCD Subtraction   ***************
                ;
                ;       This routine performs a 2 Digit Unsigned BCD Subtraction.
                ; It is assumed that the two BCD numbers to be subtracted are in
                ; locations Num_1 & Num_2. The result is the difference of Num_1 & Num_2
                ; ( Num_2 - Num_1) and is stored in location Num_2 and the overflow carry
                ; is returned in location Num_1.
                ;
                ;   Performance :
                ;                Program Memory  :       31
                ;                Clock Cycles    :       21   ( worst case )
                ;
                ;*****************************************************************
                ;
                      LIST    P=16C54
0008            Num_1   equ     8       ; Overflow flow carry overwrites Num_1
0008            result  equ     8
                ;
0009            Num_2   equ     9       ; Num_2 - Num_1 overwrites Num_2
0009            O_flow  equ     9
                ;
                  include         "mpreg.h"
                ;*************************       PIC16C5X Header ************************
01FF            PIC54   equ     1FFH    ; Define Reset Vectors
01FF            PIC55   equ     1FFH
03FF            PIC56   equ     3FFH
07FF            PIC57   equ     7FFH
                ;
0001            RTCC    equ     1h
0002            PC      equ     2h
0003            STATUS  equ     3h      ; F3 Reg is STATUS Reg.
0004            FSR     equ     4h
                ;
0005            Port_A  equ     5h
0006            Port_B  equ     6h      ; I/O Port Assignments
0007            Port_C  equ     7h
                ;
                ;*****************************************************************
                ;
                ;                       ; STATUS REG. Bits
0000            CARRY   equ     0h      ; Carry Bit is Bit.0 of F3
0000            C       equ     0h
0001            DCARRY  equ     1h
0001            DC      equ     1h
0002            Z_bit   equ     2h      ; Bit 2 of F3 is Zero Bit
0002            Z       equ     2h
0003            P_DOWN  equ     3h
0003            PD      equ     3h
0004            T_OUT   equ     4h
0004            TO      equ     4h
0005            PA0     equ     5h
0006            PA1     equ     6h
0007            PA2     equ     7h
                ;
0001            Same    equ     1h
                ;
0000            LSB     equ     0h
0007            MSB     equ     7h
                ;
0001            TRUE    equ     1h
0001            YES     equ     1h
0000            FALSE   equ     0h
0000            NO      equ     0h
                ;
                ;*****************************************************************
```


© 1994 Microchip Technology Inc.

```
                        ;
0000 0208               BCDSub  movf    Num_1,w
0001 00A9                       subwf   Num_2
0002 0068                       clrf    Num_1
0003 0368                       rlf     Num_1
0004 0723                       btfss   STATUS,DC
0005 0A0C                       goto    adjst1
0006 0769                       btfss   Num_2,3         ; Adjust LSD of Result
0007 0A0E                       goto    Over_1
0008 0649                       btfsc   Num_2,2
0009 0A0C                       goto    adjst1          ; Adjust LSD of Result
000A 0729                       btfss   Num_2,1
000B 0A0E                       goto    Over_1          ; No : Go for MSD
000C 0C06               adjst1  movlw   6
000D 00A9                       subwf   Num_2
000E 0708               Over_1  btfss   Num_1,0         ; CY = 0 ?
000F 0A17                       goto    adjst2          ; Yes, adjust MSD of result
0010 0068                       clrf    Num_1
0011 07E9                       btfss   Num_2,7         ; No, test for MSD >9
0012 0800                       RETLW   0
0013 06C9                       btfsc   Num_2,6
0014 0A17                       goto    adjst2
0015 07A9                       btfss   Num_2,5
0016 0800                       RETLW   0
0017 0C60               adjst2  movlw   60              ; add 6 to MSD
0018 00A9                       subwf   Num_2
0019 0068                       clrf    Num_1
001A 0703                       btfss   STATUS,CARRY    ; test if underflow
001B 0800                       RETLW   0
001C 0C01                       movlw   1
001D 0028                       movwf   Num_1
001E 0800               Over    RETLW   0
                        ;
                        ;*****************************************************************
                        ;               Test Program
                        ;*****************************************************************
001F 0C23               main    movlw   23
0020 0028                       movwf   Num_1           ; Set Num_1 = 23
0021 0C99                       movlw   99
0022 0029                       movwf   Num_2           ; Set Num_2 = 99
0023 0900                       call    BCDSub          ; After subtraction, Num_2 = 76 ( 99-23 )
                        ;                               ;  and Num_1 = 0 ( indicates positive result
                        ;
0024 0C99                       movlw   99
0025 0028                       movwf   Num_1           ; Set Num_1 = 99
0026 0C00                       movlw   0
0027 0029                       movwf   Num_2           ; Set Num_2 = 0
                        ;
0028 0900                       call    BCDSub          ; After subtraction, Num_2 = 1
                        ;                               ;  and Num_1 = 1 ( indicates negative result
                        ;                               ;  -1  <-  ( -99 )
                        ;
0029 0A29               self    goto    self
                        ;
                              org     1FF
01FF 0A1F                      goto    main
                        ;
                              END


Errors   :     0
Warnings :     0
```

© 1994 Microchip Technology Inc.

APPENDIX K: SQUARE ROOT BY NEWTON-RAPHSON METHOD

```
MPASM 01.01 Released   SQRT_94.ASM   8-4-1994  10:38:49          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001
                     0002         list    p=16C54, n=0
                     0003 ;*****************************************************************
                     0004 ;
                     0005 ;       Program:    SQRT.ASM
                     0006 ;       Revision:   08-03-94
                     0007 ;
                     0008 ;
                     0009 ;              Square Root By Newton Raphson Method
                     0010 ;
                     0011 ;    This routine computes the square root of a 16 bit number(with
                     0012 ;  low byte in NumLo & high byte in NumHi ). After loading NumLo &
                     0013 ;  NumHi with the desired number whose square root is to be computed,
                     0014 ;  branch to location Sqrt ( by "GOTO  Sqrt" ). " CALL  Sqrt" cannot
                     0015 ;  be issued because the Sqrt function makes calls to Math routines
                     0016 ;  and the stack is completely used up.
                     0017 ;       The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                     0018 ;  The total number of iterations is set to ten. If more iterations
                     0019 ;  are desired, change "LupCnt equ .10" to the desired value. Also,
                     0020 ;  the initial guess value of the square root is given set as
                     0021 ;  input/2 ( in subroutine "init" ). The user may modify this scheme
                     0022 ;  if a better initial approximation value is known. A good initial
                     0023 ;  guess will help the algorithm converge at a faster rate and thus
                     0024 ;  less number of iterations required.
                     0025 ;       Two utility math routines are used by this program : D_divS
                     0026 ;  and D_add. These two routines are listed as seperate routines
                     0027 ;  under double precision Division and double precision addtion
                     0028 ;  respectively.
                     0029 ;
                     0030 ;  Note : If square root of an 8 bit number is desired, it is probably
                     0031 ;         better to have a table look scheme rather than using numerical
                     0032 ;         methods.
                     0033 ;
                     0034 ;
                     0035 ;
                     0036 ;   Performance :
                     0037 ;                Program Memory  :      27  (excluding Math Routines
                     0038 ;                                        D_divS & D_add )
                     0039 ;                Clock Cycles    :    3600 ( approximately )
                     0040 ;
                     0041 ;       To assemble this program, one routine, namely "D_add"
                     0042 ;       must be included into this program. These two routines
                     0043 ;       are listed as a separate program, in files "DBL_ADD.ASM"
                     0044 ;       The D_divS routine was previously found in "DBL_DIVS.ASM".
                     0045 ;
                     0046 ;*****************************************************************
                     0047     include "picreg.h"
                     0047
                     0048         org     0
                     0049 ;
000A                 0050 LupCnt  equ     .10             ; Number of iterations
                     0051 ;
001D                 0052 NumLo   equ     1D
001E                 0053 NumHi   equ     1E
001F                 0054 count   equ     1F
                     0055 ;
                     0056 ; These are used by the divide routine
                     0057 ;
0010                 0058 ACCaLO  equ     10
0011                 0059 ACCaHI  equ     11
0012                 0060 EXPa    equ     12
```


© 1994 Microchip Technology Inc.

```
0013                    0061 ACCbLO  equ     13
0014                    0062 ACCbHI  equ     14
0015                    0063 EXPb    equ     15
0016                    0064 ACCcLO  equ     16
0017                    0065 ACCcHI  equ     17
0018                    0066 ACCdLO  equ     18
0019                    0067 ACCdHI  equ     19
001A                    0068 temp    equ     1A
001B                    0069 sign    equ     1B
                        0070 ;
                        0071 ;
0010                    0072 SqrtLo  equ     ACCaLO
0011                    0073 SqrtHi  equ     ACCaHI
                        0074 ;
                        0075 init
0000 0C0A               0076         movlw   LupCnt
0001 003F               0077         movwf   count
0002 021E               0078         movf    NumHi,W
0003 0031               0079         movwf   SqrtHi
0004 021D               0080         movf    NumLo,W         ; set initial guess root = NUM/2
0005 0030               0081         movwf   SqrtLo
0006 0403               0082         bcf     STATUS,CARRY
0007 0331               0083         rrf     SqrtHi
0008 0330               0084         rrf     SqrtLo
0009 0800               0085         retlw   0
                        0086 ;
000A 0403               0087 div2    bcf     STATUS,CARRY
000B 0314               0088         rrf     ACCbHI,W
000C 0031               0089         movwf   SqrtHi
000D 0313               0090         rrf     ACCbLO,W
000E 0030               0091         movwf   SqrtLo
000F 0800               0092         retlw   0
                        0093 ;
0010 0900               0094 Sqrt    call    init
0011 021D               0095 sloop   movf    NumLo,W
0012 0033               0096         movwf   ACCbLO
0013 021E               0097         movf    NumHi,W
0014 0034               0098         movwf   ACCbHI
                        0099 ;
0015 091B               0100         call    D_divS          ; double precision division
0016 0946               0101         call    D_add           ; double precision addition
                        0102 ;
0017 090A               0103         call    div2
0018 02FF               0104         decfsz  count
0019 0A11               0105         goto    sloop
001A 0A52               0106         goto    over            ; all iterations done
                        0107 ;                               ; branch back to desired location
                        0108 ;
                        0109 ; Double precision routines placed here
                        0110 ;
                        0111 ;*****************************************************************
0000                    0112 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
                        0113 ;                               ; for Multiplication & Division needs
                        0114 ;                               ; to be assembled as Signed Integer
                        0115 ;                               ; Routines. If 'FALSE' the above two
                        0116 ;                               ; routines ( D_mpy & D_div ) use
                        0117 ;                               ; unsigned arithmetic.
                        0118 ;*****************************************************************
                        0119 ;                       Double Precision Division
                        0120 ;
                        0121 ;               ( Optimized for Code Size : Looped Code )
                        0122 ;
                        0123 ;*****************************************************************;
                        0124 ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
                        0125 ;                                              Remainder in ACCc (16 bits
                        0126 ;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
                        0127 ;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
                        0128 ;       (c) CALL D_div
                        0129 ;       (d) The 16 bit result is in location ACCbHI & ACCbLO
```


© 1994 Microchip Technology Inc.

```
                0130 ;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
                0131 ;
                0132 ;   Performance :
                0133 ;                Program Memory  :    037
                0134 ;                Clock Cycles    :    310
                0135 ;
                0136 ;        NOTE :
                0137 ;                The performance specs are for Unsigned arithmetic ( i.e,
                0138 ;                with "SIGNED equ  FALSE ").
                0139 ;
                0140 ;*****************************************************************;
                0141 ;
                0142 ;       Double Precision Divide ( 16/16 -> 16 )
                0143 ;
                0144 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                0145 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc (ACCcHI,ACCcLO).
                0146 ;
                0147 ;   NOTE  :  Before calling this routine, the user should make sure that
                0148 ;            the Numerator(ACCb) is greater than Denominator(ACCa). If
                0149 ;            the case is not true, the user should scale either Numerator
                0150 ;            or Denominator or both such that Numerator is greater than
                0151 ;            the Denominator.
                0152 ;
                0153 ;
                0154 D_divS
                0155 ;
                0156      IF   SIGNED
                0157         CALL    S_SIGN
                0158      ENDIF
                0159 ;
001B 0937       0160          call    setup
001C 0077       0161          clrf    ACCcHI
001D 0076       0162          clrf    ACCcLO
001E 0403       0163 dloop    bcf     STATUS,CARRY
001F 0378       0164          rlf     ACCdLO
0020 0379       0165          rlf     ACCdHI
0021 0376       0166          rlf     ACCcLO
0022 0377       0167          rlf     ACCcHI
0023 0211       0168          movf    ACCaHI,w
0024 0097       0169          subwf   ACCcHI,w        ;check if a>c
0025 0743       0170          btfss   STATUS,Z_bit
0026 0A29       0171          goto    nochk
0027 0210       0172          movf    ACCaLO,w
0028 0096       0173          subwf   ACCcLO,w        ;if msb equal then check lsb
0029 0703       0174 nochk    btfss   STATUS,CARRY    ;carry set if c>a
002A 0A32       0175          goto    nogo
002B 0210       0176          movf    ACCaLO,w        ;c-a into c
002C 00B6       0177          subwf   ACCcLO
002D 0703       0178          btfss   STATUS,CARRY
002E 00F7       0179          decf    ACCcHI
002F 0211       0180          movf    ACCaHI,w
0030 00B7       0181          subwf   ACCcHI
0031 0503       0182          bsf     STATUS,CARRY    ;shift a 1 into b (result)
0032 0373       0183 nogo     rlf     ACCbLO
0033 0374       0184          rlf     ACCbHI
0034 02FA       0185          decfsz  temp            ;loop untill all bits checked
0035 0A1E       0186          goto    dloop
                0187 ;
                0188      IF   SIGNED
                0189          btfss   sign,MSB        ; check sign if negative
                0190          retlw   0
                0191          goto    neg_B           ; negate ACCa ( -ACCa -> ACCa )
                0192      ELSE
0036 0800       0193          retlw   0
                0194      ENDIF
                0195 ;
                0196 ;*****************************************************************
                0197 ;
0037 0C10       0198 setup    movlw   .16             ; for 16 shifts
```


© 1994 Microchip Technology Inc.

```
0038 003A           0199          movwf   temp
0039 0214           0200          movf    ACCbHI,w          ;move ACCb to ACCd
003A 0039           0201          movwf   ACCdHI
003B 0213           0202          movf    ACCbLO,w
003C 0038           0203          movwf   ACCdLO
003D 0074           0204          clrf    ACCbHI
003E 0073           0205          clrf    ACCbLO
003F 0800           0206          retlw   0
                    0207 ;
                    0208 ;*****************************************************************
                    0209 ;
0040 0270           0210 neg_A    comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
0041 02B0           0211          incf    ACCaLO
0042 0643           0212          btfsc   STATUS,Z_bit
0043 00F1           0213          decf    ACCaHI
0044 0271           0214          comf    ACCaHI
0045 0800           0215          retlw   0
                    0216 ;
                    0217 ;*****************************************************************
                    0218 ;  Assemble this section only if Signed Arithmetic Needed
                    0219 ;
                    0220      IF    SIGNED
                    0221 ;
                    0222 S_SIGN   movf    ACCaHI,W
                    0223          xorwf   ACCbHI,W
                    0224          movwf   sign
                    0225          btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                    0226          goto    chek_A
                    0227 ;
                    0228          comf    ACCbLO          ; negate ACCb
                    0229          incf    ACCbLO
                    0230          btfsc   STATUS,Z_bit
                    0231          decf    ACCbHI
                    0232          comf    ACCbHI
                    0233 ;
                    0234 chek_A   btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                    0235          retlw   0
                    0236          goto    neg_A
                    0237 ;
                    0238      ENDIF
                    0239 ;
                    0240     include  <D_add.asm>
                    0001 ;*****************************************************************
                    0002 ;                 Double Precision Addition & Subtraction
                    0003 ;
                    0004 ;          Program:   D_ASM.ASM
                    0005 ;          Revision:  08-03-94
                    0006 ;
                    0007 ;*****************************************************************;
                    0008 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                    0009 ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                    0010 ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                    0011 ;       (c) CALL D_add
                    0012 ;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                    0013 ;
                    0014 ;   Performance :
                    0015 ;               Program Memory  :       07
                    0016 ;               Clock Cycles    :       08
                    0017 ;*****************************************************************
                    0018 ;       Double Precision  Addition ( ACCb + ACCa -> ACCb )
                    0019 ;
0046 0210           0020 D_add    movf    ACCaLO,w
0047 01F3           0021          addwf   ACCbLO          ;add lsb
0048 0603           0022          btfsc   STATUS,CARRY    ;add in carry
0049 02B4           0023          incf    ACCbHI
004A 0211           0024          movf    ACCaHI,w
004B 01F4           0025          addwf   ACCbHI            ;add msb
004C 0800           0026          retlw   0
                    0027 ;
```

5

© 1994 Microchip Technology Inc.

```
                    0028
                    0029
                    0240
                    0241 ;
                    0242 ;*************************************************************
                    0243 ;                Test Program
                    0244 ;*************************************************************
                    0245 ;
004D 0CF3           0246 main    movlw   0F3
004E 003E           0247         movwf   NumHi
004F 0CF6           0248         movlw   0F6        ; Set input test number = 62454
0050 003D           0249         movwf   NumLo      ; = F3F6h
                    0250 ;
0051 0A10           0251         goto    Sqrt       ; cannot use CALL : Math routines
                    0252 ;                          ; use up all the stack.
0052 0000           0253 over    nop                ; all iterations done
                    0254 ;
0053 0A53           0255 self    goto    self       ; result = 00F9h = 249
                    0256 ;                          ; exact sqrt(62454) = 249.9
                    0257 ;
                    0258         org     PIC54
01FF 0A4D           0259         goto    main
                    0260 ;
                    0261         END
                    0262


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXX---------- ---------- ----------

0180 : ---------- ---------- ---------- ----------
01C0 : ---------- ---------- ---------- ----------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

APPENDIX L: INCLUDE FILE FOR FIXED POINT ROUTINE

```
                processor       16C71

;       define assembler constants

B0              equ     0
B1              equ     1
B2              equ     2
B3              equ     3
B4              equ     4
B5              equ     5

MSB             equ     7
LSB             equ     0

W               equ     0

;       define special function registers

                cblock 0x00             ; page 0 registers
                       INDF,RTCC,PCL,STATUS,FSR,TRISA,TRISB,ZZZZ,
                       ADCON0,ADRES,PCLATH,INTCON
                endc

                cblock 0x00             ; page 1 registers
                       INDF,OPTION,PCL,STATUS,FSR,PORTA,PORTB,ZZZZ,
                       ADCON1,ADRES,PCLATH,INTCON
                endc

;        define beginning of general purpose RAM

RAMSTART        equ     0x0C
RAMSTOP equ     0x2F

;       define commonly used bits

;       STATUS bit definitions
                #define _C              STATUS,0
                #define _DC             STATUS,1
                #define _Z              STATUS,2
                #define _PD             STATUS,3
                #define _TO             STATUS,4
                #define _RP0            STATUS,5
                #define _PA0            STATUS,5
                #define _RP1            STATUS,6
                #define _PA1            STATUS,6
                #define _IRP            STATUS,7
                #define _PA2            STATUS,7
```

5

© 1994 Microchip Technology Inc.

APPENDIX M: 16/8 FIXED POINT DIVIDE ROUTINES

```
;       16/8 PIC16 FIXED POINT DIVIDE ROUTINES  VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD1507U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine            Clocks     Function

;       FXD1608S    188 16 bit/8 bit -> 16.08 signed fixed point divide

;       FXD1608U    294 16 bit/8 bit -> 16.08 unsigned fixed point divide

;       FXD1607U    174 16 bit/7 bit -> 16.07 unsigned fixed point divide

;       FXD1507U    166 15 bit/7 bit -> 15.07 unsigned fixed point divide

;       The above timings are based on the looped macros. If space permits,
;       approximately 41-50 clocks can be saved by using the unrolled macros.

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       16/08 BIT Division Macros


SDIV1608L       macro

;       Max Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks
```


© 1994 Microchip Technology Inc.

```
;       Min Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       PM: 42                                  DM: 5

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                MOVLW           6
                MOVWF           LOOPCOUNT

LOOPS1608A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1608A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS1608B      RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1608B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1608L  macro

;       Max Timing: 2+7*12+11+3+7*24+23 = 291 clks

;       Min Timing: 2+7*11+10+3+7*17+16 = 227 clks

;       PM: 39                                  DM: 7
```


© 1994 Microchip Technology Inc.

```
                MOVLW           8
                MOVWF           LOOPCOUNT

LOOPU1608A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK68A
                ADDWF           REM+B0
                BCF             _C
UOK68A          RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1608A

                CLRF            TEMP

                MOVLW           8
                MOVWF           LOOPCOUNT

LOOPU1608B      RLF             ACC+B1,W
                RLF             REM+B0
                RLF             TEMP
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSS           _C
                INCFSZ          ACC+B5,W
                SUBWF           TEMP

                BTFSC           _C
                GOTO            UOK68B
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSC           _C
                INCFSZ          ACC+B5,W
                ADDWF           TEMP

                BCF             _C
UOK68B          RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1608B

                endm

UDIV1607L       macro

;       Max Timing:     7+6*11+10+10+6*11+10+2 = 171 clks

;       Min Timing:     7+6*11+10+10+6*11+10+2 = 171 clks

;       PM: 39                                  DM: 5

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1607A      RLF             ACC+B0,W
```


© 1994 Microchip Technology Inc.

```
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1607A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1607B      RLF           ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1607B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1507L       macro

;       Max Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       Min Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       PM: 42                                  DM: 5

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                MOVLW           6
                MOVWF           LOOPCOUNT

LOOPU1507A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
```

5

```
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1507A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1507B      RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1507B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

SDIV1608        macro

;       Max Timing:     3+5+14*8+2 = 122 clks

;       Min Timing:     3+5+14*8+2 = 122 clks

;       PM: 122                         DM: 4

                variable i

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
```


© 1994 Microchip Technology Inc.

```
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i=i+1

                endw

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1608  macro

;       restore = 9/21 clks,  nonrestore = 8/14 clks

;       Max Timing: 8*9+1+8*21 = 241 clks

;       Min Timing: 8*8+1+8*14 = 177 clks

;       PM: 241                                 DM: 6

                variable        i

                i = 0

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK68#v(i)
                ADDWF           REM+B0
                BCF             _C
UOK68#v(i)      RLF             ACC+B0

                i=i+1
```

```
                endw

                CLRF            TEMP

                i = 8

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                RLF             TEMP
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSS           _C
                INCFSZ          ACC+B5,W
                SUBWF           TEMP

                BTFSC           _C
                GOTO            UOK68#v(i)
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSC           _C
                INCFSZ          ACC+B5,W
                ADDWF           TEMP

                BCF             _C
UOK68#v(i)      RLF             ACC+B1

                i=i+1

                endw

                endm

UDIV1607        macro

;       Max Timing:     5+15*8+2 = 127 clks

;       Min Timing:     5+15*8+2 = 127 clks

;       PM: 127                                 DM: 4

                variable i

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                i = 1

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0
```


© 1994 Microchip Technology Inc.

```
                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i=i+1

                endw

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1507        macro

;       Max Timing:     3+5+14*8+2 = 122 clks

;       Min Timing:     3+5+14*8+2 = 122 clks

;       PM: 122                                 DM: 4

                variable i

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF     REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
```

5

```
                ADDWF           REM+B0
                RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF     REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i=i+1

                endw

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

;*************************************************************************************************
;*************************************************************************************************

;       16/8 Bit Signed Fixed Point Divide 16/8 -> 16.08

;       Input:  16 bit signed fixed point dividend in AARG+B0, AARG+B1
;               8 bit signed fixed point divisor in BARG+B0

;       Use:    CALL    FXD1608S

;       Output: 16 bit signed fixed point quotient in AARG+B0, AARG+B1
;               8 bit signed fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     10+163+3 = 176 clks             A > 0, B > 0
;                                  11+163+11 = 185 clks         A > 0, B < 0
;                                  14+163+11 = 188 clks         A < 0, B > 0
;                                  15+163+3 = 181 clks          A < 0, B < 0

;       Min Timing:     10+163+3 = 176 clks             A > 0, B > 0
;                                  11+163+11 = 185 clks         A > 0, B < 0
;                                  14+163+11 = 188 clks         A < 0, B > 0
;                                  15+163+3 = 181 clks          A < 0, B < 0

;       PM: 15+42+10 = 67               DM: 6

FXD1608S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                MOVWF           SIGN

                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA1608S

                COMF            BARG+B0
                INCF            BARG+B0

CA1608S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C1608S

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C1608S          CLRF            REM+B0

                SDIV1608L

                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B0
                INCF            REM+B0

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       16/8 Bit Unsigned Fixed Point Divide 16/8 -> 16.08

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1608U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+291+2 = 294 clks

;       Min Timing:     1+227+2 = 230 clks

;       PM: 1+39+1 = 41         DM: 7

FXD1608U        CLRF            REM+B0

                UDIV1608L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       16/7 Bit Unsigned Fixed Point Divide 16/7 -> 16.07

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
```

5

© 1994 Microchip Technology Inc.

```
;               7 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1607U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+171+2 = 174 clks

;       Min Timing:     1+171+2 = 174 clks

;       PM: 1+39+1 = 41         DM: 5

FXD1607U        CLRF            REM+B0

                UDIV1607L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       15/7 Bit Unsigned Fixed Point Divide 15/7 -> 15.07

;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               7 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1507U

;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+163+2 = 166 clks

;       Min Timing:     1+163+2 = 166 clks

;       PM: 1+42+1 = 44         DM: 5

FXD1507U        CLRF            REM+B0

                UDIV1507L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1994 Microchip Technology Inc.

APPENDIX N: 16/16 FIXED POINT DIVIDE ROUTINES

```
;       16/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD1515U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine             Clocks     Function

;       FXD1616S    319 16 bit/16 bit -> 16.16 signed fixed point divide

;       FXD1616U    373 16 bit/16 bit -> 16.16 unsigned fixed point divide

;       FXD1515U    294 15 bit/15 bit -> 15.15 unsigned fixed point divide

;       The above timings are based on the looped macros. If space permits,
;       approximately 65-69 clocks can be saved by using the unrolled macros.


                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Division Macros

SDIV1616L       macro

;       Max Timing:     13+14*18+17+8 = 290 clks

;       Min Timing:     13+14*16+15+3 = 255 clks

;       PM: 42                                  DM: 7
```

```
                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B1
                RLF             ACC+B0

                MOVLW           15
                MOVWF           LOOPCOUNT

LOOPS1616       RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            SADD66L

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66LL

SADD66L         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1616

                BTFSC           ACC+B1,LSB
                GOTO            SOK66L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
SOK66L

                endm

UDIV1616L       macro

;       restore = 23 clks,  nonrestore = 17 clks

;       Max Timing:     2+15*23+22 = 369 clks

;       Min Timing:     2+15*17+16 = 273 clks

;       PM: 24                                  DM: 7

                MOVLW           16
                MOVWF           LOOPCOUNT

LOOPU1616       RLF             ACC+B0,W
```


© 1994 Microchip Technology Inc.

```
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK66LL
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

                BCF             _C

UOK66LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1616

                endm

UDIV1515L       macro

;       Max Timing:     13+14*18+17+8 = 290 clks

;       Min Timing:     13+14*17+16+3 = 270 clks

;       PM: 42                                  DM: 7

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B1
                RLF             ACC+B0

                MOVLW           15
                MOVWF           LOOPCOUNT

LOOPU1515       RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            UADD55L

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55LL

UADD55L         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
```


© 1994 Microchip Technology Inc.

```
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1515

                BTFSC           ACC+B1,LSB
                GOTO            UOK55L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK55L

                endm

SDIV1616        macro

;       Max Timing:     7+10+6*14+14+7*14+8 = 221 clks

;       Min Timing:     7+10+6*13+13+7*13+3 = 202 clks

;       PM: 7+10+6*18+18+7*18+8 = 277   DM: 6

                variable i

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            SADD66#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66#v(i)
```


© 1994 Microchip Technology Inc.

```
SADD66#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66#v(i)      RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            SADD668

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK668

SADD668         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK668          RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            SADD66#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66#v(i)

SADD66#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66#v(i)      RLF             ACC+B1

                i=i+1

                endw

                BTFSC           ACC+B1,LSB
                GOTO            SOK66
                MOVF            BARG+B1,W
```

5

© 1994 Microchip Technology Inc.

```
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
SOK66

                endm

UDIV1616  macro

;       restore = 20 clks,  nonrestore = 14 clks

;       Max Timing: 16*20 = 320 clks

;       Min Timing: 16*14 = 224 clks

;       PM: 16*20 = 320         DM: 6

                variable         i

                i = 0

                while i < 16

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

                BCF             _C

UOK66#v(i)      RLF             ACC+B1
                RLF             ACC+B0


                i=i+1

                endw

                endm

UDIV1515        macro

;       Max Timing:     7+10+6*14+14+7*14+8 = 221 clks

;       Min Timing:     7+10+6*13+13+7*13+3 = 202 clks

;       PM:     7+10+6*18+18+7*18+8 = 277       DM: 6

                variable i

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            UADD55#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55#v(i)

UADD55#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55#v(i)      RLF             ACC+B0

                i=i+1

                endw
;***********************************************************************
;***********************************************************************
                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            UADD558

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK558

UADD558         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
```

5

© 1994 Microchip Technology Inc.

```
UOK558          RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            UADD55#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55#v(i)

UADD55#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55#v(i)      RLF             ACC+B1

                i=i+1

                endw

                BTFSC           ACC+B1,LSB
                GOTO            UOK55
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK55

                endm

;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16

;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               16 bit fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1616S

;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               16 bit fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     11+290+3 = 304 clks             A > 0, B > 0
;                               15+290+14 = 319 clks            A > 0, B < 0
;                               15+290+14 = 319 clks            A < 0, B > 0
;                               19+290+3 = 312 clks             A < 0, B < 0

;       Min Timing:     11+255+3 = 269 clks             A > 0, B > 0
;                               15+255+14 = 284 clks            A > 0, B < 0
;                               15+255+14 = 284 clks            A < 0, B > 0
;                               19+255+3 = 277 clks             A < 0, B < 0
```


© 1994 Microchip Technology Inc.

```
;       PM: 19+42+13 = 74               DM: 8

FXD1616S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN
                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA1616S

                COMF            BARG+B1
                INCF            BARG+B1
                BTFSC           _Z
                DECF            BARG+B0
                COMF            BARG+B0

CA1616S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C1616S

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C1616S          CLRF            REM+B0
                CLRF            REM+B1

                SDIV1616L

                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B1
                INCF            REM+B1
                BTFSC           _Z
                DECF            REM+B0
                COMF            REM+B0

                RETLW           0x00


;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1616U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+369+2 = 373 clks

;       Min Timing:     2+273+2 = 277 clks

;       PM: 2+24+1 = 27         DM: 7

FXD1616U        CLRF            REM+B0
```

5

```
                CLRF            REM+B1

                UDIV1616L

                RETLW           0x00
;*****************************************************************************************
;*****************************************************************************************

;       15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15

;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1515U

;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+290+2 = 294 clks

;       Min Timing:     2+270+2 = 274 clks

;       PM: 2+42+1 = 45         DM: 7

FXD1515U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV1515L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1994 Microchip Technology Inc.

APPENDIX O: 32/16 FIXED POINT DIVIDE ROUTINES

```
;       32/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD3115U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine            Clocks     Function

;       FXD3216S    578 32 bit/16 bit -> 32.16 signed fixed point divide

;       FXD3216U    702 32 bit/16 bit -> 32.16 unsigned fixed point divide

;       FXD3115U    541 31 bit/15 bit -> 31.15 unsigned fixed point divide

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Division Macros

SDIV3216L       macro

;       Max Timing:     9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks

;       Min Timing:     9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks

;       PM: 157                                 DM: 9

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
```

```
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216A      RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            SADD26LA

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LA

SADD26LA        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LA         RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216A
                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            SADD26L8

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L8

SADD26L8        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L8         RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216B      RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            SADD26LB

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
```


© 1994 Microchip Technology Inc.

```
                SUBWF           REM+B0
                GOTO            SOK26LB

SADD26LB        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LB         RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216B

                RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            SADD26L16

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L16

SADD26L16       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            SADD26LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LC

SADD26LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216C

                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
```

5

© 1994 Microchip Technology Inc.

```
                GOTO            SADD26L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L24

SADD26L24       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L24        RLF             ACC+B3

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216D      RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B3,LSB
                GOTO            SADD26LD

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LD

SADD26LD        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LD         RLF             ACC+B3

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216D

                BTFSC           ACC+B3,LSB
                GOTO            SOK26L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
SOK26L

                endm

UDIV3216L       macro

;       Max Timing:     15+6*22+21+21+6*22+21+21+6*22+21+21+6*22+21+8 = 698 clks

;       Min Timing:     15+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 662 clks

;       PM: 233                                 DM: 11

                CLRF            TEMP

                MOVF            BARG+B1,W
                SUBWF           REM+B1
```


© 1994 Microchip Technology Inc.

```
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                SUBWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSS   _C
                INCFSZ  SIGN,W
                SUBWF   TEMP
                RLF     ACC+B0

                MOVLW   7
                MOVWF   LOOPCOUNT

LOOPU3216A      RLF     ACC+B0,W
                RLF     REM+B1
                RLF     REM+B0
                MOVF    BARG+B1,W
                BTFSS   ACC+B0,LSB
                GOTO    UADD26LA

                SUBWF   REM+B1
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                SUBWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSS   _C
                INCFSZ  SIGN,W
                SUBWF   TEMP
                GOTO    UOK26LA

UADD26LA        ADDWF   REM+B1
                MOVF    BARG+B0,W
                BTFSC   _C
                INCFSZ  BARG+B0,W
                ADDWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSC   _C
                INCFSZ  SIGN,W
                ADDWF   TEMP

UOK26LA         RLF     ACC+B0

                DECFSZ  LOOPCOUNT
                GOTO    LOOPU3216A

                RLF     ACC+B1,W
                RLF     REM+B1
                RLF     REM+B0
                MOVF    BARG+B1,W
                BTFSS   ACC+B0,LSB
                GOTO    UADD26L8

                SUBWF   REM+B1
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                SUBWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSS   _C
                INCFSZ  SIGN,W
                SUBWF   TEMP
                GOTO    UOK26L8

UADD26L8        ADDWF   REM+B1
```

5

© 1994 Microchip Technology Inc.

```
                MOVF    BARG+B0,W
                BTFSC   _C
                INCFSZ  BARG+B0,W
                ADDWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSC   _C
                INCFSZ  SIGN,W
                ADDWF   TEMP

UOK26L8         RLF     ACC+B1

                MOVLW   7
                MOVWF   LOOPCOUNT

LOOPU3216B      RLF     ACC+B1,W
                RLF     REM+B1
                RLF     REM+B0
                MOVF    BARG+B1,W
                BTFSS   ACC+B1,LSB
                GOTO    UADD26LB

                SUBWF   REM+B1
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                SUBWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSS   _C
                INCFSZ  SIGN,W
                SUBWF   TEMP
                GOTO    UOK26LB

UADD26LB        ADDWF   REM+B1
                MOVF    BARG+B0,W
                BTFSC   _C
                INCFSZ  BARG+B0,W
                ADDWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSC   _C
                INCFSZ  SIGN,W
                ADDWF   TEMP

UOK26LB         RLF     ACC+B1

                DECFSZ  LOOPCOUNT
                GOTO    LOOPU3216B

                RLF     ACC+B2,W
                RLF     REM+B1
                RLF     REM+B0
                MOVF    BARG+B1,W
                BTFSS   ACC+B1,LSB
                GOTO    UADD26L16

                SUBWF   REM+B1
                MOVF    BARG+B0,W
                BTFSS   _C
                INCFSZ  BARG+B0,W
                SUBWF   REM+B0
                CLRF    SIGN
                CLRW
                BTFSS   _C
                INCFSZ  SIGN,W
                SUBWF   TEMP
                GOTO    UOK26L16
```


© 1994 Microchip Technology Inc.

```
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

U0K26L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3216C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD26LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            U0K26LC

UADD26LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

U0K26LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3216C

                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD26L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            U0K26L24
```

5

```
UADD26L24       ADDWF       REM+B1
                MOVF        BARG+B0,W
                BTFSC       _C
                INCFSZ      BARG+B0,W
                ADDWF       REM+B0
                CLRF        SIGN
                CLRW
                BTFSC       _C
                INCFSZ      SIGN,W
                ADDWF       TEMP

UOK26L24        RLF         ACC+B3

                MOVLW       7
                MOVWF       LOOPCOUNT

LOOPU3216D      RLF         ACC+B3,W
                RLF         REM+B1
                RLF         REM+B0
                MOVF        BARG+B1,W
                BTFSS       ACC+B3,LSB
                GOTO        UADD26LD

                SUBWF       REM+B1
                MOVF        BARG+B0,W
                BTFSS       _C
                INCFSZ      BARG+B0,W
                SUBWF       REM+B0
                CLRF        SIGN
                CLRW
                BTFSS       _C
                INCFSZ      SIGN,W
                SUBWF       TEMP
                GOTO        UOK26LD

UADD26LD        ADDWF       REM+B1
                MOVF        BARG+B0,W
                BTFSC       _C
                INCFSZ      BARG+B0,W
                ADDWF       REM+B0
                CLRF        SIGN
                CLRW
                BTFSC       _C
                INCFSZ      SIGN,W
                ADDWF       TEMP

UOK26LD         RLF         ACC+B3

                DECFSZ      LOOPCOUNT
                GOTO        LOOPU3216D

                BTFSC       ACC+B3,LSB
                GOTO        UOK26L
                MOVF        BARG+B1,W
                ADDWF       REM+B1
                MOVF        BARG+B0,W
                BTFSC       _C
                INCF        BARG+B0,W
                ADDWF       REM+B0
UOK26L

                endm

UDIV3115L       macro

;       Max Timing:     9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks

;       Min Timing:     9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks
```


© 1994 Microchip Technology Inc.

```
;       PM: 157                                 DM: 9

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115A      RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD15LA

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LA

UADD15LA        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LA         RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115A

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD15L8

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15L8

UADD15L8        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15L8         RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115B      RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
```

5

© 1994 Microchip Technology Inc.

```
                GOTO            UADD15LB

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LB

UADD15LB        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LB         RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115B

                RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            UADD15L16

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15L16

UADD15L16       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD15LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LC

UADD15LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115C
```


© 1994 Microchip Technology Inc.

```
                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD15L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15L24

UADD15L24       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15L24        RLF             ACC+B3

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115D      RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B3,LSB
                GOTO            UADD15LD

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LD

UADD15LD        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LD         RLF             ACC+B3

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115D

                BTFSC           ACC+B3,LSB
                GOTO            UOK15L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK15L

                endm

;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16

;       Input:  32 bit fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
```

5

```
;               16 bit fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3216S

;       Output: 32 bit fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     11+537+3 = 551 clks             A > 0, B > 0
;                                  15+537+20 = 572 clks         A > 0, B < 0
;                                  21+537+20 = 578 clks         A < 0, B > 0
;                                  25+537+3 = 565 clks          A < 0, B < 0

;       Min Timing:     11+501+3 = 515 clks             A > 0, B > 0
;                                  15+501+20 = 536 clks         A > 0, B < 0
;                                  21+501+20 = 542 clks         A < 0, B > 0
;                                  25+501+3 = 529 clks          A < 0, B < 0

;       PM: 25+157+19 = 201             DM: 10

FXD3216S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN
                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA3216S

                COMF            BARG+B1
                INCF            BARG+B1
                BTFSC           _Z
                DECF            BARG+B0
                COMF            BARG+B0

CA3216S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C3216S

                COMF            AARG+B3
                INCF            AARG+B3
                BTFSC           _Z
                DECF            AARG+B2
                COMF            AARG+B2
                BTFSC           _Z
                DECF            AARG+B1
                COMF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C3216S          CLRF            REM+B0
                CLRF            REM+B1

                SDIV3216L

                BTFSS           SIGN,MSB        ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B3
                INCF            AARG+B3
                BTFSC           _Z
                DECF            AARG+B2
                COMF            AARG+B2
                BTFSC           _Z
                DECF            AARG+B1
                COMF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B1
```


© 1994 Microchip Technology Inc.

```
                INCF            REM+B1
                BTFSC           _Z
                DECF            REM+B0
                COMF            REM+B0

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16

;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3216U

;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+698+2 = 702 clks

;       Max Timing:     2+662+2 = 666 clks

;       PM: 2+233+1 = 236               DM: 11

FXD3216U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV3216L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15

;       Input:  31 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3115U

;       Output: 31 bit unsigned fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+537+2 = 541 clks

;       Min Timing:     2+501+2 = 505 clks

;       PM: 2+157+1 = 160               DM: 9

FXD3115U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV3115L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```

5

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

AN585

A Real-Time Operating System for PIC16/17

Author: Jerry Farmer - Myriad Development Company

I. INTRODUCTION

Ever dream of having a Real-Time Kernel for the PIC16CXX family of microprocessors ? Or ever wonder what Multitasking or Threads were all about? Then this article is for you. We will explore HOW to implement all of the features of a large Real-Time Multitasking Kernel in much less space, with more control, without the large overhead of existing kernels. By planning ahead, and using the techniques outlined here, you can build your own fast, light, powerful, flexible RT kernels with just the features needed to get the job done.

Included in this article are two large examples: one on the PIC16C54, and the other on the new powerful PIC16C64. A "Remote Alarm" is implemented on the PIC16C54 as an example of a Non-Preemptive Kernel, with two asynchronous serial input sources capable of running up to 19,200 Baud along with seven sensors needing to be debounced as inputs. One more input line is monitored and causes an internal software recount. For output, this example has a LED that show eight different internal states of the "Remote Alarm", blinking at different rates and different sequences. Last but not least, is an asynchronous Serial output capable of running at 38,400 Baud passing the inputs to the next remote alarm station. Several short and long timers are included to round out the nine cooperating tasks in this example. Please refer to Figure 2 and Appendix B.

The second example is implemented on an PIC16C64 featuring an interrupt driven semi-Preemptive kernel. This example has the serial input and output routines of the first example moved into Interrupt Service Routines (ISR) for more speed and accuracy. The interrupt capabilities of the PIC16C64 will be explored, and a Real-Time Multitasking Kernel framework will be developed. Please refer to Figure 5 and Appenidx C.

Why do I Need a Real-Time Kernel ?

Real-Time Design Techniques allow the engineer/designer to break-up large, complicated problems into smaller simpler tasks or threads. These more manageable units of code allow for faster response to important events, while prioritizing the jobs to be done in a structured well tested format. The kernel does the job of keeping the time, the peace between tasks, and keeping all the tasks' communication flowing. More activities can be performed in the same amount of time by allowing other tasks to work while other tasks are waiting for some event to occur. Smaller code is also the result of using State-Driven techniques because much information is condensed into the state variables and code structure. If you need an example, look at the PIC16C54's "Remote Alarm" code.

What is Multitasking Anyway ?

This is the appearance of several tasks working at the same time. Each task thinks that it owns the CPU, but this appearance is controlled by the kernel. Only one task can be running at a time, but there is undone work that can be done by other tasks not blocked. Multitasking is the orchestration of interrupts, events, communication, shared data, and timing to get a job done. Real-Time Programming is just a bunch of ideas, concepts and techniques that allow us to divide problems into units of code that are based on units of time, or events that drive a task from one state to another.

© 1994 Microchip Technology Inc.

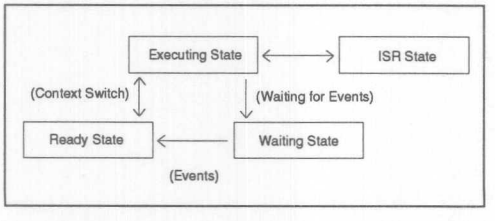
CONCEPTS

We will cover the basic concepts of kernels here so that we are using the same language when talking about this difficult topic. This article is a very quick survey on Real-Time Kernel concepts. I hope to get you thinking, reading more, and hopefully writing RT Operating Systems for your current and future projects. Many great books have been written about this very broad and interesting subject. We will refer to some of these books which have different points of view other than expressed in this paper.

Critical Section

A critical section is a shared data structure, or a shared resource, or a critical time section of code, or a non-reentrant section of code that can have only one owner that is allowed to view/change/use that section at any one time. These sections must not be interrupted during the update process. These sections must be protected so that other tasks can not get in and change the pointers/data or modify the hardware at the same time. Remember that if two tasks can get into a critical section, then data WILL be corrupted. Make sure that critical sections are small, with time for pending interrupts to get serviced. Not understanding critical sections is where the beginning RT programmers get into the most trouble. Even without interrupts, you must protect variables that are changing over time such as the byte sized variable "xmt_byte" used in the PIC16C54 example. This variable changes each time the STATE changes for the Serial Out Task. Semaphores, and Disabling Interrupts are some of the techniques used to coordinate between different tasks wanting to control a critical section. Task #4 is devoted to the proper feeding of the shared Serial Out Resource in the PIC16C54 example. Note the use of the binary semaphore "OState_B" to control Task #4, and Task #1, and variable "xmt_byte". There are several more examples of critical sections in the PIC16C64 example because of interrupts. We disable interrupts for very short time periods to protect these areas. Also in the PIC16C64 example, all critical sections are finished before checking to see if the kernel wants another task to start running instead of the current task. We will discuss in more detail how to protect critical sections later in this article.

FIGURE 1: TASK / PROCESS STATE TRANSITION DIAGRAM



Shared Resources

Data structures, displays, I/O hardware, non-reentrant routines are a few resource examples. If two or more tasks use these resources, then they are called Shared Resources and you must protect them from being corrupted. They must have only one owner, a way of telling others to wait, and maybe a waiting list for future users of that resource. A rare example of a shared resource is when there exists a critical timing sequence of input and output operations to control some hardware. You must disable interrupts before starting this sequence, and enable them upon finishing. Note that task #1 in the PIC16C64 example is an example of an "non-reentrant" routine that must be finished by the current owner before another task can use it.

Context Switch/Task Switch

When one task takes over from another, the current values of the CPU registers for this running task are saved and the old saved registers for the new task are restored. The new task continues where it left off. This is all done by the Context Switch part of the Real-Time Kernel. Each task usually has a "context switch storage area". Each task's SP (Stack Pointer pointing into its own stack) is stored there along with all the other important saved registers. The "Remote Alarm" example does not need to use a context switch because all the important registers are properly freed-up before each task is finished. The PIC16C64 example uses a similar concept, thus keeping the number of saved registers per task way down. We use an old concept called "where I came from". The variable "FROM" is used to direct the dispatcher to start up the task where it left off. This is because you can not manipulate the stack in the PIC16CXX family. This same reason is why we have a "semi- Preemptive" kernel on the PIC16C64 as an example. By the way, the faster the Context Switch is done, the better.

Scheduler

The scheduler is that part of the kernel that decides which task to run next. We will talk about several common types here in this section. This is where a lot of thinking should be done before starting your new project. By understanding the different kinds of schedulers and what features and problems each type has, you can match your problem to a creatively styled scheduler that meets your needs. For example, the PIC16C54 example shows the recalling of tasks #1-3 just before a long sequence of code is executed. More creative ways can also be done, but be careful to allow all tasks to execute in a timely fashion.

Please see Figure 1. Each task must be in "Ready State" or the "Executing State" to be considered by the scheduler to get temporary control of the CPU next.

© 1994 Microchip Technology Inc.

NON-PREEMPTIVE KERNEL

The Non-Preemptive Kernel is also called a "Cooperative Kernel" because the tasks only give-up control when they want/need to in coordination with other tasks, and events. The "Remote Alarm" example uses a Non-Preemptive Kernel type, showing that despite its reputation as being a simple kernel type, a lot can be done with it. The Non-Preemptive Kernel type is well suited for the non-interrupt type PIC16CXXs. The heart beat of the PIC16C54 example is the internal RTCC counter crossing over from high value to low value of the counter. Use the pre-scaler to adjust the time units. The very fast tasks, continually read the RTCC directly, comparing the delta of time to see if it should fire.

PREEMPTIVE KERNEL

In a Preemptive Kernel, a running task can be swapped out for a higher priority task when it becomes ready. The Preemptive Kernel relies much more on interrupts as its driving force. The context switch is at the heart of this type of kernel. To implement a true Preemptive Kernel, you must be able to manipulate the stack. This is why we implemented a "semi-Preemptive" kernel on the PIC16C64, with some of the best features of both types of kernels. We moved some of the tasks in the PIC16C54 example into ISRs to handle the I/Os. This works very well as the ISRs are very short and do most of the real work in this example. The timer0 interrupt is the heart beat for the PIC16C64 example. You must have a clock interrupt in order to make a true Preemptive kernel.

ROUND ROBIN SCHEDULER

When the scheduler finds tasks on the ready queue that have the same priorities, the scheduler often uses a technique called Round Robin scheduling to make sure each task gets its day in the sun. This means more housekeeping to get it right. This is part of the creative ways you can tailor the scheduler to fit your needs. In the PIC16C54 example, all tasks will get to run shortly after their appointed time. This means that no task will dominate all others in this simple approach. In the "olden" days of the first Real-Time Operating Systems the term was used to mean the same as "time slicing". The Preemptive Kernels of today are a major step forward, with their priority schemes, and intertask communication capabilities.

PREEMPTIVE VS. NON-PREEMPTIVE

The Preemptive Kernel is the harder to develop, but is easier to use, and is sometimes used incorrectly. You must spend more upfront time with the Non-Preemptive Kernel but it is better for more cramped microprocessors. You get much better response time between a cause/event and the response/action for that event with a Non-Preemptive Kernel. The Preemptive Kernel is more predictable in the response times, and can be calculated as to the maximum time to complete a given job. Often the Preemptive Kernel is more expensive.

REENTRANCY

In a Preemptive Kernel, two or more tasks may want to use the same subroutine. The problem is that you can not control when a task is swapped out and when another takes its place. Thus, if a subroutine uses only local or passed variables that are stored only in each tasks' stack, then it is call reentrant or a pure routine. No global variables or hardware may be used in such a pure routine. A way around this reentrancy requirement is to treat the whole subroutine as a critical section. Appendix D is an example of reentrant code segment as might have been used in the PIC16C54 code example.

Task Priority

Some tasks are not created equal. Some jobs must be done on time or data will be lost. Make the tasks that must get done, the highest priority, and go down the scale from there. Some kernels make you have a different priority for each task. This is a good idea and requires some thought before coding to make the design come out.

STATIC VS. DYNAMIC PRIORITIES AND PRIORITY INVERSIONS

For most embedded Real-Time Kernels, both static priorities and static tasks are used. Dynamic priorities are sometimes used to solve deadlock and other complex situations that arises from not understanding the problem and not understanding Real-Time Techniques. You should relook at how you divided the problem, and divide less so as to include the resources in question under one semaphore. You could divide the problem more to have more tasks not needing two or more resources to complete its job, and have the new tasks talk more together.

© 1994 Microchip Technology Inc.

As for Dynamic tasks, you should define the problem so as to know ahead of coding, the continuous use of all task. You will need more upfront time in the planning stage to get all tasks talking, but it is well worth it to keep Dynamic Priorities and Dynamic Tasking out of the kernel design.

Priority Inversions is a trick used to get passed a poorly designed system by allowing the inverting of the priorities to allow lower tasks to run that were blocked before. This is a cheap trick, and is best kept out of a Real-Time Kernel. Use the same techniques as outlined in this section to solve this kind of problem.

Semaphores

There are basically two types: binary and counting semaphores. The binary semaphore is used to allow only one owner, and all other tasks wanting access are made to wait. The counting semaphore, keeps a list of users that want to get access. Semaphores can be used in many ways, we will illustrate most of them in the following paragraphs. Note that you can implement counting semaphores using binary semaphores.

MUTUAL EXCLUSION

We have discussed Mutual Exclusion before, that there must be a way to exclude other tasks from gaining access to critical sections. Mutual Exclusion is the process of excluding others from access to the shared resources. To make a semaphore is a very complicated process. The semaphore's construction must be atomic. That means that once the process has started, it can not be interrupted until it has saved the name of the new owner. From there on, it knows that no one else can break-in and change owners. We have implemented the binary semaphore using bits and kernel functions to mutually exclude access in the PIC16C54 example.

In the PIC16C64 example, we also disable interrupts to get the same effect. There are at least two good ways of implementing a binary semaphore. The first and oldest way was discovered by a Dutch mathematician named Dekker. We will refer you to a book that talks more about this algorithm. The second method of implementing a binary semaphore was also discovered by another Dutchman named Dijkstra. This method uses the "testandset" type instruction and is much more important to us. We used the "dec & jump if not zero" instruction (see PIC16C64 example).

DEADLOCK

Deadlock is a condition where two or more tasks own resources that other tasks need to complete there assignment but will not release their own resources until the other tasks release theirs. Talk about cooperation. Please read section, "Static vs. Dynamic Priorities and Priority Inversions" for a discussion about such problems and ways to solve them. The root of such problems is not understanding the original problem.

SYNCHRONIZATION

Semaphores can be used to synchronize tasks so that messages can be passed between them. Also tasks can be started up by semaphores, stopped by semaphores, or started together. They are the foundation blocks for Real-Time Programming. Once you have built a binary semaphore for your kernel, you can build very complex semaphores to synchronize anything. In the PIC16C54 example, data from several sources are passed out the Serial Port Resource. Task #4 synchronizes the other tasks trying to send data out and synchronizes with task #1 to get it done. When task #1 is running, then task #4 can not run until task #1 is ready for more data to send out.

INTERTASK COMMUNICATION

We have touched on this topic already, but for large kernels, one can include more complex communication methods to pass data/messages between tasks. Much of the handshaking is done for you inside the kernel. This takes a lot more space and execution speed to implement them in a kernel.

Event Flags

We implemented Event Flags as simple bits having two states (on and off). More info can be stored per Event Flag such as time it was recorded, by who, and who the event belongs to, and whether data was lost.

Message Mailboxes

This is a nice feature to have if you have the ram space. Mailboxes allow the designer to pass messages between tasks, and allows messages to be looked at when the task is ready, and to reply telling the sender that the message was received. One message can be sent to many tasks at the same time.

Message Queues

This again is a very nice feature if you have the execution time, and the ram to implement them. This feature is related to Mailboxes, in that you can store several messages even after reading, to be processed later. If you want to only operate on the highest prioritized messages before handling the rest, this is allowed. You can be very fancy with the Mailboxes and Queues. If you have them, use them.

© 1994 Microchip Technology Inc.

Interrupts

Interrupts are one of the best inventions to come along for solving Real-Time problems. You can get very quick response to the need, and then go back to what you were doing. The only problem is that they can and do happen at the worst times. That means that you must learn how to turn them on and off to protect your critical sections. Note that before an interrupt can be handled, you must save all important registers so that you can restore them so that the kernel can restart the task where it left off. This is much like the context switch issue, but for interrupts, you must always save and restore. In the PIC16C64 example, the Status, W, and FSR registers are saved in ram because of the interrupt. The PC register is saved onto the stack by hardware.

INTERRUPT LATENCY, RESPONSE & RECOVERY

Interrupt Latency is defined as the largest time period that interrupts are disabled, plus the time it takes for the ISR to start to execute.

The Interrupt Response Time is defined for a Non-Preemptive system as Interrupt Latency plus the "context saving time". For a Preemptive system, add the execution time for the kernel to record the interrupt.

Interrupt Recovery Time for a Non-Preemptive system is defined as the time to restore the saved context and for the restarting of the task that was interrupted. Interrupt Recovery Time for a Preemptive system is the same as for the Non-Preemptive system plus the time the kernel takes in the scheduler deciding which task to run next. These measurements are how most kernels are compared with each other. The PIC16C64 example does very well in these measurements. That is because of the PIC16CXX processor and that they are mostly a Non-Preemptive system. You must keep the time you disable interrupts to a minimum in any kernel you write or any task that you write. You should break-up long sequences of instructions to allow for interrupts that are already waiting to execute.

ISR PROCESSING TIME

ISR (Interrupt Service Routine) Processing Time is defined as the time a ISR keeps control of the CPU. This amount of time should be short, and if alot of processing needs to be done in a ISR, then break up the ISR. The new ISR should now just store the new data and return. Next, create a new task and move the extra code from the old ISR into the new task. Remember that the longer you are in one interrupt, the longer you can not answer another pressing interrupt.

Nesting interrupts are where the interrupt with a higher priority can interrupt a lower priority interrupt. Care must be used, as different interrupts may have critical sections too, and disabling interrupts must be used here too to protect critical sections. Nesting of interrupts may not exist on all microprocessors, such as the PIC16CXX family.

NON-MASKABLE INTERRUPTS

On some microprocessors, you can enable/disable selected interrupts, such as on the PIC16/17 family. This is a great tool to control the flow of data into the system and out. Some systems have what is called Non-Maskable Interrupts. Here you can not turn them off by software masking. These NMIs as they are call for short, are used as clock Ticks, because you do not want problems with complex critical sections on a interrupt that you can not turn off. The PIC16CXX family does not have any NMIs. NMIs are not as useful as maskable interrupts.

CLOCK TICK

The Clock Tick, is the heart beat of the system. This is how the kernel keeps time (relative & absolute). This is how the kernel is restarted to see if there is a delay that has finished, so that that task can be moved into the ready state. In the PIC16C54 example, the RTCC clock is used. In the PIC16C64 example, the timer0 is used. You must have a clock interrupt in order to make a true Preemptive kernel. This is the other reason why we implemented a Non-Preemptive Kernel on the PIC16C54 - no clock interrupt.

© 1994 Microchip Technology Inc.

ANALYSIS OF CODE EXAMPLES

These sections are the real meat of this article. In these sections we will explain how the concepts are put to practical use line by line in each of the two main examples - PIC16C54 (Appendix C) and PIC16C64 (Appendix D).

We will also examine a short reentrant code example in Appendix B. We will give some ideas on how to expand the examples and how far and how fast the examples can be pushed. Be sure to read both sections on the two examples.

The "Remote Alarm" application has many interesting features. The concept is to have as many tiers of units like a tree feeding into the lower level units the status of each of the larger branches to one central point. Each unit can detect any changes in status before the intruder shuts that unit down, or tampers with it. If any unit's power or wires connecting it down the tree are cut, the lack of the flow of status and passwords would be noticed in five seconds and reported down the line. The two Serial Input lines per unit receive the status and passwords from it's two larger branches, checking the data and passing the info down the line by its own Serial Output line. The seven input status lines are debounced in these examples, showing the technique.

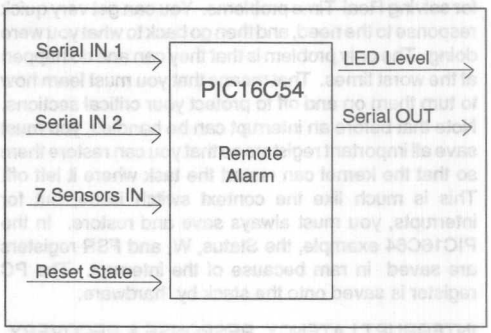
The LED on each unit reports the status at that node as to the importance of its own seven input status lines and the status flowing down the line. The level indication outputted on the LED continues at the last highest level until either a reset is received on the "Reset State" line or five minutes of no new activity on the seven input status lines are received. When either of these two events occur, the level of the LED output is adjusted to the current level of input. Some of the features are changed for this article. See Figures 2 and 5.

Another Embedded System use of this type of "Remote Alarm" application is that of placing the unit on the outside of a safe. Hopefully the intruder would be detected before arriving at the unit itself. The continuous stream of status and passwords to the larger unit inside would slow down any simple theft.

PIC16C54 - "Remote Alarm" Example

This example is a cross between a true application and an attempt to show new concepts and some extra features for show. Some of the application specific code has been removed to show more clearly the possibilities of a Real_Time Operating System on the PIC16/17 family. We chose the Baud rate for the Serial output to be twice the speed of the two Serial inputs because it is harder to accurately output a precise Serial Output than it is to monitor Serial inputs.

FIGURE 2: REMOTE ALARM - PIC16C54 EXAMPLE



This example operates at 4Mhz. By simply increasing the crystal speed to 8MHz, the two Asynchronous input Serial Baud rates increase from 4800 Baud to 9600 Baud. The Serial Output Baud rate increases from 9600 Baud to 19,200 Baud. By increasing the crystal speed to 16MHz, it will increase the Baud rates to 19,200 Baud for the two independent Asynchronous inputs, and increase the baud rate for the Asynchronous Serial output to 38,400 Baud. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds. Note that you must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C54 code example. Lines 1-85 are the equates for this program. Lines 88-95 are simple jump tables so as to save some of the precious "first 256 bytes" of each page. The Serial Output Routines - Task #1 are in lines 97-159. Task #7's subroutines start at line 160 and continue to line 277. In this section, the LED output is controlled. The subroutine QCheck_T123, lines 278-301, is used to allow the checking of Tasks #1-3 to see if they are ready to execute before a long section of code in a slower Task is about to be executed. This is a creative way for the Kernel's Scheduler to makes sure that the highest Prioritized Tasks get serviced before the less important tasks get executed. Task #2 starts at line 302. This task reads the Serial Input #1 for Asynchronous data. Task #2 can be described as a State Machine for outputting a byte Serially. Task #3 interrupts the code of Task #2 at line 333 and continues until line 362. Task #3 also reads the Serial Input but on input #2. Task #2's subroutines continue at line 363 and continue until line 423. Task #3's subroutines continue at line 424 and continue until line 484 is reached. The main or starting code is started at line 485. From that line to line 515, all variables are initialized, and all tasks are initialized at this time also.

© 1994 Microchip Technology Inc.

The Main Loop is started at line 516 and ends at line 665. This is where the real action is done. Each task checks the time to see if theconditions are correct for it to run. The tasks that are not Blocked, andhave a job to do now are in a Ready State. In the Main Loop, we check the current state of each task in order of Priority (1-9). If ready, we do a very simple Task Switch and place that task in the Executing State/Running State. Several time unit changes take place in the Main Loop. Tasks #1-4 use 2 uSecs as a time base by reading the RTCC directly. A time unit change takes place at lines 562-575 to 512 uSecs per unit for Tasks #5-6. Another time unit change takes place for Tasks #7-9, to 131072 uSecs per unit. For Tasks #5-9, each task counts the time units and compares them to their standard for activation or activity. Task #4 starts at line 538 and finishes at line 561. Task #4 controls the feeding of Task #1 from several other tasks that want data to be outputted. It uses several Semaphores to make sure that Task #1 is not bothered until it is ready for another byte. Task #5 monitors the Level Reset Line, and is always running. It simply resets the status of the LED, to be recalculated in Task #6. Task #5 runs through lines 576-581, and is very short. Lines 582-611 represent Task #6. Here we debounce the seven sensor input lines, leaving the current standard in the variable "Old_RB". Task #6 asks/Signals Task #4 to output the current standard out the Serial pin. Task #7's main code is lines 621-628. Task #8 is a five second lack of activity timer, and exists in lines 629-645. If no data comes from either of the two input Serial lines, then Task #8 Signals Task #4 to send a special byte to be outputted by Task #1. This Signals the next "Remote Alarm" of the lack of communication between units. The last task is Task #9. This is a five minute lack of Severe Errors the from Sensor Reset Timer. Lines 646-663 compose Task #9. Subroutine Do_D_H_E_L starts at line 667 and continues through to line 692. This routine determines the Highest Error Level, and passes Task #7, the current state, to output on the LED. Lines 693-703, clear the registers #7-1Fh. The "jump at Power-On" code is the last lines 705-706.

The following sections describe in more detail how and what each part of the code does and why. The code segment lines 1-87 are explained in this paragraph. Line 4 tells the MPASM assembler which PIC16/17 you are using. The include file "PICREG.H" follows with the equates and assignments to make the code more readable and changeable. You should use equates that relate symbols to each other. The Constants - lines 10-12 are the values to change for different Baud rates. They represent the Bit Times for the Baud rates divided by 2 minus some latency factor. You might have to adjust the "Fudge Factor" and other values to fine tune the performance. The value used for the "Fudge Factor" is related to the longest path of code. Lines 21-24 are an experiment that allows a simple name to be associated to a single bit. This allows for easily changeable assignments. Lines 30-54 are the variable assignments. Variables (lines 35-39) are used as time counters. They count the number of units of time, and are compared to literals to see if an Event has just happened. The bits defined in lines 57-64 are used as Binary Semaphores. They keep Critical Sections of data protected. We will see them in action later in the code. The bits defined in lines 67-73 are error flags. They define the current or last error states of the Serial routines, and whether data was lost coming in or out. The section of equates in lines 76-85 are used to define the different LED activity. They are used by Task #7 to keep the LED blinking. In lines 89-94, we try to save the all important first 256 bytes of any page.

Task #1 outputs a byte Asynchronously over the Serial Output pin. Task #1 is started at line 98. The time units used for Tasks #1-4 are 2µS. We first sample the RTCC and store the count. When Tasks #1-4 are then allowed to run, they check the difference between the first sample and the current time. If the delta is greater than or equal to the delay, then that Event has just happened. We first check if the state of the Serial Output is zero. We then jump to OStateS to start the outputting of the "Start Bit". Because any Serial Output timings must be rock solid, we use a trick in lines 101-116 that helps greatly. We check if we are within a certain amount of time BEFORE the deadline and then wait for the time to output another bit. This trick allows us to be within a certain ± amount of time within the expected time to output that bit. With this code, we are about <±8% accurate for the Serial Output. You can only use this trick on the most critical tasks, and only on one. In this section of code, we are constantly checking the delta of time from the "FIRST_RTCC_O" reading and the current reading of RTCC. When we are very close to the output time, we jump to line 117. If we are not even close to the proper time, we exit back to the main loop, so we can check the other timers and tasks. Now look at Figure 4 for a description of the Output Pulses, the "Bit units of Time", and the associated state numbers. Note that the activities are spread out over time.

The timer Events help to define the different states and their associated output activities. Each Event is handled in a very short, well-defined set of code as Task #1. Lines 117-131, are a quick state jump table. You need to break all Real-Time code into very short segments - in and then out. Each segment is just a few lines long. You do your activity, save status, and increment to the next state. Notice that OState0_7 code is used several times to output all 8 bits. The state variable is used also to count the number of bits already outputted. The time to the next outputting of a bit is calculatedand is adjusted to take out the accumulation of errors in lines 151-152. We make sure of a full "Stop Bit" length in the OStateE code. In the OStateL code, we reset the OState variable to zero, and tell the world that we are not outputting now in line 157. This is important because we use that bit (OState_B) to Signal that we need to protect the variable xmt_byte that changes over several states. We also use it to Signal that we are ready for another byte to output. Look at Task #4. See how it uses this Semaphore to full advantage. We have just explained a Critical Segment variable as outlined in the theory sections of this article.

© 1994 Microchip Technology Inc.

Task #2 reads the Serial Input line 1, running at 4800 Baud. The code structure is very similar to that of Task #1. See Figure 3. Notice that there are more states than the Serial Output Task #1. Once the "Start Bit" is detected, we half step into the "Start Bit" to see if it was a "False Start" or not. We then sample and store the incoming bits to form an 8-bit byte just like Task #1. We sample the "Stop Bit" to see if it is a "Frame Error". We delay another 1/2 bit to get to the end of the "Stop Bit" if there was an "Frame Error" before resetting Task #1's state to 0. Otherwise, we reset Task #1's state to 0, and Signal that we are ready for another "Start Bit". The just received byte is stored in variable "RCV_Storage". A check is made to see if we already sent out the last received byte before clobbering the old byte with the new byte.

Task #3 reads the Serial Input line 2, running at 4800 Baud. The code structure is the same as Task #2. See Figure #3. The received byte is also put into the same storage variable as Task #2 - "RCV_Storage". When either Task #2 or Task #3 receives a valid byte, Task #8's counter is reset. You can up the Baud rate of Task #2 and 3 if you lower the output Baud rate of Task #1. Note that for reading the Serial Input Lines, you can be off by ±15% for each sampling, but not accumulatively.

Task #4 finds the next buffered byte to send out through Task #1. Task #4 also controls the order of which byte goes first over another less important byte of data. It can be said that Task #1 Blocks Task #4 from running. You can think of the Serial Output Line as a Shared Resource. The use of Semaphores here allow the Synchronization of data and actions.

Task #5 monitors the Level Reset Input Line and will reset the LED state variable if the line ever goes low. This task is always in the Ready State. This task is said to simply "pole the input line" when ever it can.

Task #6 debounces the seven sensor input lines, running every 20 mSecs. The variable "T_20_mS_CO" is incremented every 512 uSecs (Clock Tick) and is compared to the count needed to equal 20 mSecs. If it is time, the subroutine QCheck_T123 is called to see if Tasks #1-3 are in the Ready State. If any of the Tasks #1-3 are ready, they are ran and we then continue with Task #6. We compare the current value of the input Port_B to see if it stayed the same from the last reading 20 mSecs back. If the two readings are the same, then Port_B is considered to be stable and the possibly new value is placed in the variable "Old_RB" to be outputted by Task #1. The subroutine D_H_E_L is called to determine the new LED state. We then check if Task #1 was too busy to output the last sensor status byte, if so then that error is recorded.

FIGURE 3: SERIAL INPUT STATES VS. TIME DIAGRAM

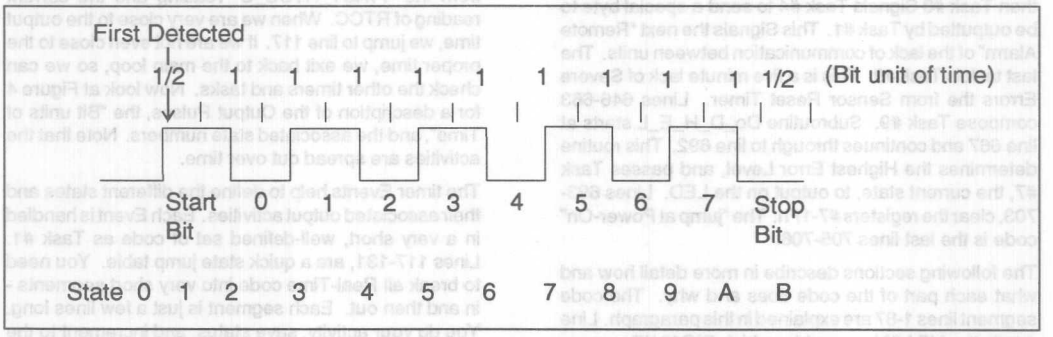
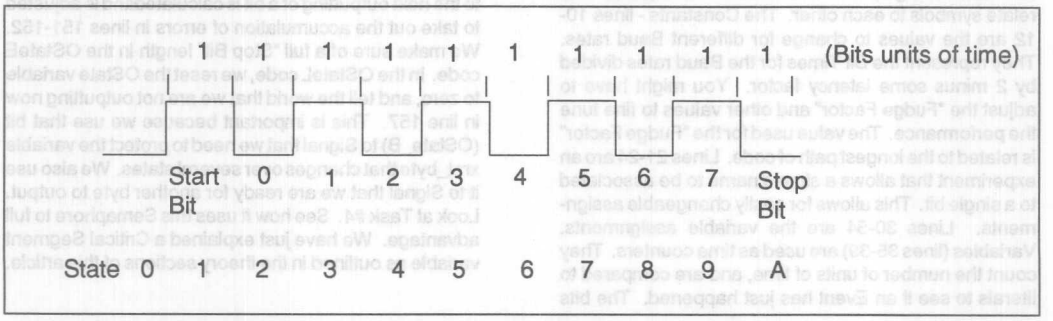


FIGURE 4: SERIAL OUTPUT STATES VS. TIME DIAGRAM



© 1994 Microchip Technology Inc.

Task #7 outputs the Highest Severity Level Indication on the LED. Do_LED starts at line 161, and continues to 276. This task is also broken into small time units of code. It is constantly checking to see if it is time to switch the on/off condition of the LED. The time units for Task #7 are regulated by the code in lines 613-619. 131072µS = time unit for Tasks #7-9. Task #7 has many state jump tables so it is included in the first 256 bytes of the first page. Lines 168-175 explain the on and off sequences and offs that represent levels of severity of the input status lines. The variable "LED_Mode" has both Task #7's current state number and the sub-state-number for that state's output sequence.

Task #8 is a 5 second lack of input from either of the two Serial input timers. Tasks #2 and #3 will reset the time counter for Task #8, when either receives a full byte. If the time counter "T_5_S_CO" equals 5 secs, then the LED's state is bumped to the highest, and a special byte is sent down the line to the next "Remote Alarm" unit. The counter variable is reset, and count starts all over. We then check if Task #1 was too busy to output the last special status byte, if so then that error is recorded.

Task #9 measures 5 minutes of calm on the 7 sensor lines and then resets the LED's state. Task #9 needs 16 bits of counter power to record 5 minutes of time. The counter variables are reset after being triggered.

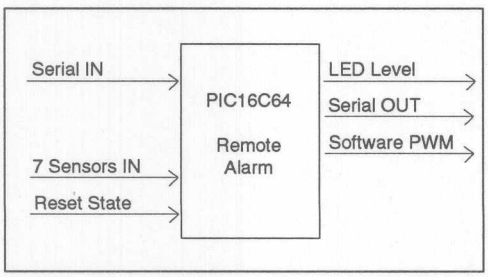
Do_D_H_E_L determines the LED's next state based on the 7 sensor input status. This subroutine checks each bit to see if it is active and then checks if a change in the LED's current state needs changing.

Do_Clear_Regs clears registers 7-1Fh. It leaves the FSR register zeroed out. This is very important for the PIC16C57 chip.

PIC16C64 - "Remote Alarm64" Example

This example is the same as the PIC16C54 example with a few changes to take advantage of the three timers on the PIC16C64 and interrupts. The second Serial input routine was replaced by an example of a software PWM (Pulse Width Modulation) example. The same code as the PIC16C54 example will run on the PIC16C64 with very few changes using only the TMR0 (RTCC). Be sure to read about the PIC16C54 example, as the comments will not be repeated, except to make a strong point.

FIGURE 5: REMOTE ALARM - PIC16C64 EXAMPLE



This example operates at 4Mhz. By simply increasing the crystal speeds, you can change the input and output Baud rates just as outlined in the section on the PIC16C54 example's crystal selection. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds. Note that you must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C64 code example. Lines 1-78 are the equates for this program. Notice that there is no need for jump tables for subroutines to be in the "first 256 bytes" of each page as there was in the PIC16C54 example. Note that the "Reset Vector" is now at code address 0, and the "Interrupt Vector" is at code address 4. Task #1 and 2 have been simplified greatly by using interrupts and timers. For Task #1, we no longer need to use the "trick" any more. It is time to execute once the routines for Task #1 and others are called. The section of code that handles the "Start Bit" (OStateS) lines 106-122 has been changed to setting up TMR2 with its interrupt to trigger the next call to this subroutine. The initial call to this subroutine was by Task #4, but later calls are due to Timer 2's interrupts. The amount of time until the next interrupt is set byeach state's code. This amount is based on the "Bit Unit of Time" which is based on Baud rate and crystal speed. An easy change to the code is to add a software selectable and "changeable on the fly" Baud rate. This is done by having a variable that selects the new Baud rate from the two data tables. One table gets you the Bit Delay value - see line 110. The other data table gets the value to be put into T2CON - see line 107, which selects the Post and Pre-scalers. You may need to adjust the Bit Delay value to take in account the Interrupt Latency. The OStateL state code shuts down Timer 2 and its interrupt. See lines 647-676 to understand how we get here by interrupt. Once Timer 2's count equals the count we put into register PR2, we get an interrupt if the following three conditions are true: 1.) Not already in an interrupt. When the current interrupt is done, our interrupt will be executed. 2.) GIE and PEIE bits are set. 3.) TMR2IE bit is set. Remember to clear the Flag bit as in line 114 before returning from an interrupt. Return from this subroutine will return you back to Task #4 or back to the ISR handle lines 647-676 depending on who called this routine. The Task #7's subroutines are the same as in the PIC16C54 example, lines 151-268. Task #2 is different from the previous example, lines 288-380. First Task #2 uses two interrupts. The INT interrupt on pin RB0/INT is used to detect the "Start Bit". It is very accurate. It is turned off after the detection in I1StateS code. The second interrupt TMR1 is then Enabled in the I1StateS code. Timer 1 is then used to cause an interrupt for all the other states for Task #2. Notice that Timer 1 has a 16-bit counter and we calculate the amount of Clock Ticks until overflow in lines 329-333. In the state code I1StateL, TMR1 is shut down, and the INT interrupt is now Enabled so as to detect the next input byte. The initializing of the PIC16C64 variable takes place in lines 383-426. The initializing of the tasks take place in lines 427-451. Notice that the last bit to be set

5

© 1994 Microchip Technology Inc.

is the GIE bit in line 451 after ALL is setup. There are several ways to execute the Task #3-9 code: by Timer 0 overflow interrupt, by having the code be in the background as in this example. The tradeoffs are many, and too deep for this article. Notice that the subroutine QCheck_T123 is not needed in this method. Timer0 overflow interrupt sets the flag: Time_Bit. The code in lines 454-457 can be considered the "IDLE Task" on some systems. It waits for a Clock Tick from TMR0's overflow. Task #3 is new, and is a simple 8-bit software PWM. Lines 459-478 show how to have 8 bits of ON, and 8 bits of OFF. This task has two states, on and off. You may add to the code by allowing the Real-Time changing of the 8-bit values under software control. When you change the values in the variables PWM_Out and PWM_In, disable all interrupts by using the following line: "BCF INTCON,GIE", and enable all interrupts by using the following line: "BSF INTCON,GIE". The new values will be used at the next transition, thus allowing a smooth change. This task could easily be used in the PIC16C54 example type Kernel. Task #4 is the same except that it calls Task #1's subroutine to initiate the outputting of a byte. See line 503. Tasks #5-9 are the same as in the PIC16C54 example. The subroutines: D_H_E_L and Clear_Regs are the same in both examples. The TMR0 (Timer 0) Overflow interrupt ISR (Interrupt Service Routine) is lines 641-645. This ISR will set the Time_Bit bit and clear the Flag that caused the interrupt. The Interrupt code lines 647-676 handles the saving of the Context Registers and the restoring of the Context Registers (W, Status, FSR) and by checking the order which interrupts are to be handled first - see lines 656-669. A very important line is 654. You must set the memory page pointers here for the ISR routines ! Line 676 is the only place that an interrupt is allowed to return and set the GIE bit (RETFIE).

Reentrant example

See Appendix E for the short code segment. This code corresponds to lines 302-332 in the PIC16C54 example. The purpose of reentrant code is to allow two or more tasks to use the same code at the "same time". See the section about reentrant in the theory section of this article. Notice how the registers 18h-1Bh match the registers 1Ch-1Fh, both starting with the state variable for the two tasks using this routine. Note how Task #2 and Task #3 load a pointer to the state variable for their task before calling DO_IState code. By using the FSR register as a pointer, and incrementing or decermenting the FSR register, you can keep the variables in the two tasks straight even if the two tasks are using different code in the subroutine at any one time. This method is not easy to implement, as can be seen, so use two copies for readability instead, like the PIC16C54 example.

SUMMARY

Now that the PIC16/17 family of microcontrollers have a way of executing Real-Time Programs, using the techniques outlined in this article, there is very little that PIC16/17s can not do! Much more than was ever dreamed before. Many of you will quickly understand and start modifying these examples. Great. That means that we have done our job at Myriad. A few of you may want more help. Great. At Myriad Development Co., we LOVE the PIC16/17 family.

© 1994 Microchip Technology Inc.

BIBLIOGRAPHY

Foster, Caxton C.
Real Time Programming - Neglected Topics
Reading, Massachusetts
Addison-Wesley Publishing Company, 1981

Holt, R.C., Graham, G.S., Lazowska, E.D., Scott, M.A.
Structured CONCURRENT PROGRAMMING with Operating Systems Applications
Reading, Massachusetts
Addison-Wesley Publishing Company, 1978

Kaisler, Stephen H.
The Design of Operating Systems for Small Computer Systems
New York, NY
John Wiley & Sons, 1983

Labrosse, Jean J.
uC/OS - The Real-Time Kernel
Lawrence, Kansas
R & D Publications, 1992

Loeliger, R.G.
Threaded Interpretive Languages
Peterborough, NH
BYTE BOOKS, 1981

5

APPENDIX A

A Real-Time Vocabulary

ASYNCHRONOUS - an activity that can happen at any moment, at any time

BLOCKING - the act of wanting to waiting for an EVENT before continuing

CLOCK TICK - the heart beat that all time is based on

CONTEXT/TASK SWITCH - module that saves and restores the states of a task

CRITICAL SECTION - section of code or hardware - only one user at a time

DEADLOCK - that is where two TASKs are waiting for each others resources

DISPATCHING - the act of starting up a TASK to run from an RT Kernel

DYNAMIC PRIORITIES - the ability for TASKs to have there PRIORITIES changed

DYNAMIC TASKING - the creation and the killing of TASKs

EMBEDDED SYSTEM - an internal system that operates all by itself

ENABLING/DISABLING INTERRUPTS - controlling the interrupting processing

EVENT - Timer, communication, handshaking, interrupts, data, external events

EVENT FLAGS - the storage of current states or info on what has happened

INTERRUPT - a hardware event (external/internal) that triggers a jump to the ISR routines to handle that event

INTERRUPT LATENCY - how long it takes once signaled to start an ISR

INTERRUPT RECOVERY - how long it takes once interrupted to return back to code

KERNEL - module that controls TASKs, INTERRUPTs, and intertask communications

MAILBOXES - a way to pass data from one TASK to another

MASKABLE INTERRUPTS - the ability to control whether an ISR is called or not

MULTITASKING - the act of several TASKs thinking they own the CPU

MUTUAL EXCLUSION - the act of allowing only ONE owner to a RESOURCE

NMI - Non-Maskable Interrupt - can not be turned off by software

READY STATE - Referring to a list of TASKs ready (having work to do NOW)

REENTRANT - code that can be used by several TASKs at the same time

RESOURCE - data structures, display, I/O hardware, non-reentrant routines

RUNNING STATE - Referring to the ONE task owning/using the CPU currently

SCHEDULER - that part of a kernel that decides which TASK to run next

SEMAPHORES - a protocol to control RESOURCES, SIGNAL EVENTS, synchronize tasks

SIGNAL - the act of one task signaling another that something has happened

STATE MACHINE - an important concept in dividing a job into TASKs & ISRs

SYNCHRONIZATION - were TASKs synchronize over data or at a special time

TASK PRIORITY - each TASK is ranked as to its importance to getting done

TASK/THREAD - code that is defined by a small coherent job/work to be done

TIME SLICING - the act of giving the same amount of "time" to each TASK to run

TRAP - a software caused interrupt, useful for system access

WAITING STATE - Referring to a list of TASKs waiting for an EVENT(s)

© 1994 Microchip Technology Inc.

APPENDIX B

```
; 'Reentrant Code Example' Designed by Myriad Development Co. - Jerry Farmer
;       PIC16C54, 4MHz Crystal, WatchDog Timer OFF

; Register Files
IState1         equ     18h             ;Serial In #1 State
First_RTCC_I1   equ     19h             ;Starting time for next #1 Input event
nbti1           equ     1Ah             ;Next Bit #1 In Time - variable time
rcv_byte_1      equ     1Bh             ;Receive Serial #1 In byte
IState2         equ     1Ch             ;Serial In #2 State
First_RTCC_I2   equ     1Dh             ;Starting time for next #2 Input event
nbti2           equ     1Eh             ;Next Bit #2 In Time - variable time
rcv_byte_2      equ     1Fh             ;Receive Serial #2 In byte


;******         ;Task 2,3 - Asynchronous 2400 Baud Serial Input (LOW=0)
Do_IState
                movf    INDIR           ;if IState2 == 0
                btsc    STATUS,Z        ;               then Do Start Bit
                goto    IStateS
                movf    RTCC,W          ;Get current time
                movwf   temp
                incf    FSR             ;Point to First_RTCC_I(1,2)
                movf    INDIR,W         ;Get elapsed time; Time Unit = 2 uS
                subwf   temp
                incf    FSR             ;Point to nbti(1,2)
                movf    INDIR,W         ;Past time for next input bit ?
                subwf   temp,W
                btss    STATUS,0
                goto    L1
L0
                movlw   2               ;Point to IState(1,2)
                subwf   FSR
                movf    INDIR,W         ;Get (0-B) mode #
                andlw   H'0F'           ;Get only mode #
                addwf   PC              ;jump to subroutine

                goto    IStateS         ;Serial Start Bit
                goto    IStateS2        ;1/2 of Start Bit - see if False Start
                goto    IState0_7       ;Bit 0
                goto    IState0_7       ;Bit 1
                goto    IState0_7       ;Bit 2
                goto    IState0_7       ;Bit 3
                goto    IState0_7       ;Bit 4
                goto    IState0_7       ;Bit 5
                goto    IState0_7       ;Bit 6
                goto    IState0_7       ;Bit 7
                goto    IStateE         ;Serial Stop Bit
                goto    IStateL         ;Last State
L1
                clrf    FSR             ;Clear the FSR register
                retlw   0

;*****
Task_2           ;Task 2 - Asynchronous 2400 Baud Serial Input (LOW=0)
                movlw   IState1         ;Point to IState1
                movwf   FSR
                call    Do_IState
;*****
Task_3           ;Task 3 - Asynchronous 2400 Baud Serial Input (LOW=0)
                movlw   IState2         ;Point to IState2
                movwf   FSR
                call    Do_IState

                END
```

© 1994 Microchip Technology Inc.

```
MPASM 01.00.01 Inter.     APP_C.ASM   7-29-1994  11:18:16              PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ; 'Remote Alarm' V1.02 Designed by Myriad Development Co. - Jerry Farmer
                     0002 ;       PIC16C54, 4MHz Crystal, WatchDog Timer OFF, MPASM instruction set
                     0003
                     0004         list    p=16C54,t=ON,c=132
                     0005
                     0006         include "PICREG.H"
                     0001 ;************************  PIC16C5X Header     *************************
01FF                 0002 PIC54   equ     1FFH    ; Define Reset Vectors
01FF                 0003 PIC55   equ     1FFH
03FF                 0004 PIC56   equ     3FFH
07FF                 0005 PIC57   equ     7FFH
                     0006 ;
0001                 0007 RTCC    equ     1
0002                 0008 PC      equ     2
0003                 0009 STATUS  equ     3       ; F3 Reg is STATUS Reg.
0004                 0010 FSR     equ     4
                     0011 ;
0005                 0012 Port_A  equ     5
0006                 0013 Port_B  equ     6       ; I/O Port Assignments
0007                 0014 Port_C  equ     7
                     0015 ;
                     0016 ;*****************************************************************
                     0017 ;
                     0018 ;                       ; STATUS REG. Bits
0000                 0019 CARRY   equ     0       ; Carry Bit is Bit.0 of F3
0000                 0020 C       equ     0
0001                 0021 DCARRY  equ     1
0001                 0022 DC      equ     1
0002                 0023 Z_bit   equ     2       ; Bit 2 of F3 is Zero Bit
0002                 0024 Z       equ     2
0003                 0025 P_DOWN  equ     3
0003                 0026 PD      equ     3
0004                 0027 T_OUT   equ     4
0004                 0028 TO      equ     4
0005                 0029 PA0     equ     5
0006                 0030 PA1     equ     6
0007                 0031 PA2     equ     7
                     0032 ;
0001                 0033 Same    equ     1
0000                 0034 W       equ     0
                     0035 ;
0000                 0036 LSB     equ     0
0007                 0037 MSB     equ     7
```


© 1994 Microchip Technology Inc.

```
                    0038 ;
0001                0039 TRUE    equ     1
0001                0040 YES     equ     1
0000                0041 FALSE   equ     0
0000                0042 NO      equ     0
                    0043 ;
                    0044 ;*****************************************************************
                    0006
                    0007
                    0008 ; Constants
0000                0009 INDIR           equ     0           ;Indirect Register
0033                0010 OUT_BIT_TIME    equ     33h         ;9600 Baud, 104uS Bit Rate
0064                0011 IN_BIT_TIME     equ     64h         ;4800 Baud, 208uS Bit Rate
0023                0012 FUDGE_TIME      equ     23h         ;Current Time within a Fudge Factor
                    0013
                    0014 ; B Register Definitions
0001                0015  #define Level_Reset     Port_B,0    ;Low will cause Past Level to reset
                    0016                                      ;RB.7 - RB.1 == Input from Sensors
00FF                0017 RB_TRIS         equ     B'11111111' ;RB TRIS at INIT State == all input
0000                0018 RB_MASK         equ     B'00000000' ;What is High/Low for RB at INIT State
                    0019
                    0020 ; A Register Definitions - Programmable Inputs
0002                0021  #define Serial_IN_1     Port_A,0    ;Serial Input #1 - 8 bits
0003                0022  #define LED             Port_A,1    ;LED Output - Level/State Indicator
0004                0023  #define Serial_Out      Port_A,2    ;Serial Output - 8 bits + passwords
0005                0024  #define Serial_IN_2     Port_A,3    ;Serial Input #2 - 8 bits
                    0025
00F9                0026 RA_TRIS         equ     B'11111001' ;RA TRIS at INIT State
0000                0027 RA_MASK         equ     B'00000000' ;What is High/Low for RA at INIT State
                    0028
                    0029 ; Register Files
0007                0030 temp            equ     07h         ;Tempary holding register - PIC16C54/56
0008                0031 Timer_Bits      equ     08h         ;Indicates which Timer(s) are Active = 1
0009                0032 Flags           equ     09h         ;Error Flags
000A                0033 LED_Mode        equ     0Ah         ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
000B                0034 OState          equ     0Bh         ;Serial Out State
000C                0035 T_5_M_LO        equ     0Ch         ;5 Min Timer Counter - Low
000D                0036 T_5_M_HI        equ     0Dh         ;5 Min Timer Counter - High
000E                0037 T_5_S_CO        equ     0Eh         ;5 Second Timer - lack of Serial Input
000F                0038 T_20_mS_CO      equ     0Fh         ;20 mS Timer - used for debouncing
0010                0039 LED_C           equ     10h         ;LED Counter
0011                0040 Last_RTCC       equ     11h         ;Last value of the RTCC
0012                0041 First_RTCC_O    equ     12h         ;Starting time for next Output event
0013                0042 xmt_byte        equ     13h         ;Serial xmit byte - destroyed in use
0014                0043 cc              equ     14h         ;256 * RTCC time
0015                0044 RCV_Storage     equ     15h         ;Long term storage of rcv_byte #1 & 2
0016                0045 Old_RB          equ     16h         ;Oldest/Master copy of RB
0017                0046 Last_RB         equ     17h         ;Last copy of RB
```

© 1994 Microchip Technology Inc.

```
0018                    0047 IState1         equ     18h             ;Serial In #1 State
0019                    0048 First_RTCC_I1   equ     19h             ;Starting time for next #1 Input event
001A                    0049 nbti1           equ     1Ah             ;Next Bit #1 In Time - variable time
001B                    0050 rcv_byte_1      equ     1Bh             ;Receive Serial #1 In byte
001C                    0051 IState2         equ     1Ch             ;Serial In #2 State
001D                    0052 First_RTCC_I2   equ     1Dh             ;Starting time for next #2 Input event
001E                    0053 nbti2           equ     1Eh             ;Next Bit #2 In Time - variable time
001F                    0054 rcv_byte_2      equ     1Fh             ;Receive Serial #2 In byte
                        0055
                        0056 ; Indicates which Timer(s) are Active = 1 & Flags
0006                    0057  #define OState_B       Timer_Bits,0    ;Serial Out Active Bit
0007                    0058  #define IState1_B      Timer_Bits,1    ;Serial IN #1 Active Bit
0008                    0059  #define IState2_B      Timer_Bits,2    ;Serial IN #2 Active Bit
0009                    0060  #define T_5_S_B        Timer_Bits,3    ;5 Second Timer Active Bit
000A                    0061  #define T_5_M_B        Timer_Bits,4    ;5 Min Timer Active Bit
000B                    0062  #define RCV_Got_One_B  Timer_Bits,5    ;Got a NEW Received byte to send out
000C                    0063  #define RB_NEW_B       Timer_Bits,6    ;Indicates a change in RB input
000D                    0064  #define S_5_S_B        Timer_Bits,7    ;Serial In 5 secs of inactivity
                        0065
                        0066 ; Error Flags
000E                    0067  #define FS_Flag_1      Flags,0         ;Serial #1 IN had a False Start Error
000F                    0068  #define FE_Flag_1      Flags,1         ;Last Serial #1 IN had a Frame Error
0010                    0069  #define FS_Flag_2      Flags,2         ;Serial #2 IN had a False Start Error
0011                    0070  #define FE_Flag_2      Flags,3         ;Last Serial #2 IN had a Frame Error
0012                    0071  #define RCV_Overflow   Flags,4         ;Lost Serial Input Byte - too Slow
0013                    0072  #define RB_Overflow    Flags,5         ;Lost RB Input Byte - too Slow
0014                    0073  #define S_5_S_Overflow Flags,6         ;Lost '5S Inactivity' msg - too Slow
                        0074
                        0075 ;Equates for LED Task #7
0015                    0076  #define LED_B          LED_Mode,3      ;LED is active
0016                    0077  #define LED_NEW_B      LED_Mode,7      ;LED has just changed Modes = 1
0008                    0078 LED_OFF_MODE    equ     B'00001000'     ;LED OFF
0089                    0079 LED_SEQ1_MODE   equ     B'10001001'     ;LED Sequence 1: .2s On, 1s Off
008A                    0080 LED_SEQ2_MODE   equ     B'10001010'     ;LED Sequence 2: 3x(.2s), 1s Off
008B                    0081 LED_SEQ3_MODE   equ     B'10001011'     ;LED Sequence 3: 5x(.2s), 1s Off
009C                    0082 LED_SLOW_MODE   equ     B'10011100'     ;LED Slow Pulsing - .3 Hz
009D                    0083 LED_MEDIUM_MODE equ     B'10011101'     ;LED Medium Pulsing - 1 Hz
009E                    0084 LED_FAST_MODE   equ     B'10011110'     ;LED Fast Pulsing - 3 Hz
008F                    0085 LED_ON_MODE     equ     B'10001111'     ;LED ON Continuously
                        0086
                        0087
                        0088 ; Clear Registers 7-1Fh
                        0089 Clear_Regs
0000 0BE9               0090         GOTO    Do_Clear_Regs   ;Save space in first 256 bytes
                        0091
                        0092 ; Determine the Highest Error Level & Start Task #7 outputing the new Level
                        0093 D_H_E_L
0001 0BD2               0094         GOTO    Do_D_H_E_L      ;Save space in first 256 bytes
```


© 1994 Microchip Technology Inc.

```
                0095
                0096 ;******                          ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
                0097 Do_OState
0002 022B       0098          MOVF    OState          ;if OState == 0
0003 0643       0099          BTFSC   STATUS,Z        ;
0004 0A24       0100          GOTO    OStateS         ;           then goto Output-Start-Bit
0005 0201       0101          MOVF    RTCC,W          ;Get current time
0006 0027       0102          MOVWF   temp            ;                   & store in tempary variable
0007 0212       0103          MOVF    First_RTCC_0,W  ;Get elapsed time; Time Unit = 2 uS
0008 00A7       0104          SUBWF   temp            ;Delta of Current Time & Orginal Time
0009 0C23       0105          MOVLW   FUDGE_TIME      ;Take in account processing time to do it
000A 0087       0106          SUBWF   temp,W          ;Time within fudge factor ?
000B 0703       0107          BTFSS   STATUS,C
000C 0A23       0108          GOTO    _0005           ;Not time yet to change States so return
000D 0C33       0109 _0003    MOVLW   OUT_BIT_TIME    ;Past time for next out-bit ?
000E 0087       0110          SUBWF   temp,W
000F 0603       0111          BTFSC   STATUS,C        ;Do some delaying until it is time
0010 0A15       0112          GOTO    _0004           ;It is now time to out put a bit
0011 0C04       0113          MOVLW   H'04'           ;Account for loop delay
0012 01E7       0114          ADDWF   temp
0013 0000       0115          NOP                     ;   make loop delay even
0014 0A0D       0116          GOTO    _0003           ;Wait for exact time to output bit
0015 020B       0117 _0004    MOVF    OState,W        ;Get (0-A) mode #
0016 0E0F       0118          ANDLW   H'0F'           ;Get only mode #
0017 01E2       0119          ADDWF   PC              ;jump to subroutine
0018 0A24       0120          GOTO    OStateS         ;Serial Start Bit
0019 0A2B       0121          GOTO    OState0_7       ;Bit 0
001A 0A2B       0122          GOTO    OState0_7       ;Bit 1
001B 0A2B       0123          GOTO    OState0_7       ;Bit 2
001C 0A2B       0124          GOTO    OState0_7       ;Bit 3
001D 0A2B       0125          GOTO    OState0_7       ;Bit 4
001E 0A2B       0126          GOTO    OState0_7       ;Bit 5
001F 0A2B       0127          GOTO    OState0_7       ;Bit 6
0020 0A2B       0128          GOTO    OState0_7       ;Bit 7
0021 0A31       0129          GOTO    OStateE         ;Serial Stop Bit
0022 0A36       0130          GOTO    OStateL         ;Last State
0023 0800       0131 _0005    RETLW   H'00'
                0132
                0133 OStateS
0024 0545       0134          BSF     Serial_Out      ;Serial Start Bit
0025 0201       0135          MOVF    RTCC,W          ;Store starting time
0026 0032       0136          MOVWF   First_RTCC_0
0027 0C0D       0137          MOVLW   H'0D'           ;Fudge again
0028 00B2       0138          SUBWF   First_RTCC_0
0029 02AB       0139          INCF    OState          ;increment to next state
002A 0800       0140          RETLW   H'00'
                0141
                0142 OState0_7                        ;Bit 0 - 7
```

© 1994 Microchip Technology Inc.

```
002B 0333  0143          RRF     xmt_byte        ;Move bit into C from right most bit
002C 0703  0144          BTFSS   STATUS,C        ;
002D 0445  0145          BCF     Serial_Out      ;
002E 0603  0146          BTFSC   STATUS,C        ;
002F 0545  0147          BSF     Serial_Out      ;
0030 0A32  0148          GOTO    OS_End
           0149 OStateE
0031 0445  0150          BCF     Serial_Out      ;Serial Stop Bit
0032 0C33  0151 OS_End   MOVLW   OUT_BIT_TIME    ;Adjust out the cumulation of error
0033 01F2  0152          ADDWF   First_RTCC_O
0034 02AB  0153          INCF    OState          ;increment to next state
0035 0800  0154          RETLW   H'00'
           0155 OStateL
0036 006B  0156          CLRF    OState          ;Ready to send next byte out
0037 0408  0157          BCF     OState_B        ;Serial Out not active
0038 0800  0158          RETLW   H'00'
           0159
           0160 ;******  ;Task #7 - Output Highest Level Indication on LED
           0161 Do_LED
0039 06EA  0162          BTFSC   LED_NEW_B       ;Initialize regs if change in modes
003A 0A4C  0163          GOTO    LED_NEW
003B 02B0  0164          INCF    LED_C           ;Inc Counter - Time Unit = 131072 uS
003C 020A  0165          MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 0E07  0166          ANDLW   H'07'           ;Get only mode #
003E 01E2  0167          ADDWF   PC              ;jump to subroutine
003F 0A48  0168          GOTO    LED_OFF         ;LED OFF
0040 0A64  0169          GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 0A67  0170          GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 0A8A  0171          GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 0A50  0172          GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 0A5E  0173          GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 0A61  0174          GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 0A4D  0175          GOTO    LED_ON          ;LED ON Continuously
0047 0800  0176 _0012    RETLW   H'00'
           0177 ;---
           0178 LED_OFF
0048 0425  0179          BCF     LED             ;Turn off LED
0049 046A  0180          BCF     LED_B           ;LED must be off
004A 0070  0181          CLRF    LED_C           ;Reset Counter - LED_C = 0
004B 0800  0182          RETLW   H'00'
           0183 ;---
           0184 LED_NEW
004C 04EA  0185          BCF     LED_NEW_B       ;Done initializing
           0186 LED_ON
004D 0525  0187          BSF     LED             ;Turn on LED
004E 0070  0188          CLRF    LED_C           ;Reset Counter - LED_C = 0
004F 0800  0189          RETLW   H'00'
           0190 ;---
```


© 1994 Microchip Technology Inc.

```
                0191 LED_SLOW
0050 0C0C       0192        MOVLW   H'0C'        ;.3Hz @ 50% Duty
0051 0027       0193        MOVWF   temp
0052 0207       0194 LED_S  MOVF    temp,W       ;Check LED_C if time, .3Hz @ 50% Duty
0053 0090       0195        SUBWF   LED_C,W
0054 0743       0196        BTFSS   STATUS,Z
0055 0A47       0197        GOTO    _0012
0056 0C10       0198        MOVLW   H'10'
0057 01AA       0199        XORWF   LED_Mode     ;Switch states
0058 078A       0200        BTFSS   LED_Mode,4   ;Now make LED same state
0059 0425       0201        BCF     LED
005A 068A       0202        BTFSC   LED_Mode,4
005B 0525       0203        BSF     LED
005C 0070       0204        CLRF    LED_C        ;Reset LED_C
005D 0800       0205        RETLW   H'00'
                0206 ;--
                0207 LED_MEDIUM
005E 0C04       0208        MOVLW   H'04'        ;1Hz @ 50% Duty
005F 0027       0209        MOVWF   temp
0060 0A52       0210        GOTO    LED_S        ;Go do it
                0211 ;--
                0212 LED_FAST
0061 0C01       0213        MOVLW   H'01'        ;3Hz @ 50% Duty
0062 0027       0214        MOVWF   temp
0063 0A52       0215        GOTO    LED_S        ;Go do it
                0216 ;--
                0217 LED_SEQ1                    ;.2 ON, 1 OFF
0064 078A       0218        BTFSS   LED_Mode,4   ;Skip if bit is high
0065 0A76       0219        GOTO    ON1          ;Go do it
0066 0A82       0220        GOTO    OFF3         ;Go do it
                0221 ;--
                0222 LED_SEQ2                    ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 020A       0223        MOVF    LED_Mode,W
0068 0027       0224        MOVWF   temp
0069 0C30       0225        MOVLW   H'30'        ;Get sequence # only
006A 0167       0226        ANDWF   temp
006B 03A7       0227        SWAPF   temp         ;swap nibbles
006C 0207       0228        MOVF    temp,W       ;get nibble for offset
006D 01E2       0229        ADDWF   PC           ;Table jump calculation
006E 0A76       0230        GOTO    ON1          ;LED is on, check if time to change
006F 0A7C       0231        GOTO    OFF2         ;LED is off, check if time to change
0070 0A76       0232        GOTO    ON1          ;LED is on, check if time to change
0071 0A82       0233        GOTO    OFF3         ;LED is off, check if time to change
                0234 ;--
                0235 LED_Exit
0072 0C10       0236        MOVLW   H'10'        ;Inc Seq #
0073 01EA       0237        ADDWF   LED_Mode
0074 0070       0238        CLRF    LED_C        ;Reset LED_C
```

© 1994 Microchip Technology Inc.

```
0075 0800          0239          RETLW   H'00'
                   0240 ON1
0076 0C02          0241          MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
0077 0090          0242          SUBWF   LED_C,W
0078 0743          0243          BTFSS   STATUS,Z
0079 0A47          0244          GOTO    _0012
007A 0425          0245          BCF     LED             ;Turn off LED
007B 0A72          0246          GOTO    LED_Exit
                   0247 OFF2
007C 0C02          0248          MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0090          0249          SUBWF   LED_C,W
007E 0743          0250          BTFSS   STATUS,Z
007F 0A47          0251          GOTO    _0012
0080 0525          0252          BSF     LED             ;Turn on LED
0081 0A72          0253          GOTO    LED_Exit
                   0254 OFF3
0082 0C08          0255          MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0090          0256          SUBWF   LED_C,W
0084 0743          0257          BTFSS   STATUS,Z
0085 0A47          0258          GOTO    _0012
0086 0525          0259          BSF     LED             ;Turn on LED
0087 0CF0          0260          MOVLW   H'F0'
0088 012A          0261          IORWF   LED_Mode        ;Cause (Seq# & NEW) to overflow to 0
0089 0A72          0262          GOTO    LED_Exit
                   0263 LED_SEQ3                         ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 020A          0264          MOVF    LED_Mode,W      ;Get LED info
008B 0027          0265          MOVWF   temp
008C 0C70          0266          MOVLW   H'70'           ;Get sequence # only
008D 0167          0267          ANDWF   temp
008E 03A7          0268          SWAPF   temp            ;swap nibbles
008F 0207          0269          MOVF    temp,W          ;get nibble for offset
0090 01E2          0270          ADDWF   PC              ;Table jump calculation
0091 0A76          0271          GOTO    ON1             ;LED is on check if time to change
0092 0A7C          0272          GOTO    OFF2            ;LED is off check if time to change
0093 0A76          0273          GOTO    ON1             ;LED is on check if time to change
0094 0A7C          0274          GOTO    OFF2            ;LED is off check if time to change
0095 0A76          0275          GOTO    ON1             ;LED is on check if time to change
0096 0A82          0276          GOTO    OFF3            ;LED is off check if time to change
                   0277
                   0278 ;****          Quick Check of Tasks #1, #2 and #3
                   0279 QCheck_T123
                   0280          ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0097 0708          0281          BTFSS   OState_B        ;if not outputing now then skip call
0098 0A9A          0282          GOTO    T2
0099 0902          0283          CALL    Do_OState       ;Go Do Task #1
                   0284
                   0285          ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
009A 0628          0286 T2       BTFSC   IState1_B       ;if already started then call
```


© 1994 Microchip Technology Inc.

```
009B 0A9F          0287          GOTO    _0029
009C 0605          0288          BTFSC   Serial_IN_1     ;if Start bit ? then call
009D 0A9F          0289          GOTO    _0029
009E 0AA0          0290          GOTO    T3
009F 09A7          0291 _0029    CALL    Do_I1State      ;Go Do Task #2
                   0292
                   0293          ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
00A0 0648          0294 T3       BTFSC   IState2_B       ;if already started then call
00A1 0AA5          0295          GOTO    _0031
00A2 0665          0296          BTFSC   Serial_IN_2     ;if Start bit ? then call
00A3 0AA5          0297          GOTO    _0031
00A4 0800          0298          RETLW   H'00'
00A5 09C2          0299 _0031    CALL    Do_I2State      ;Go Do Task #3
00A6 0800          0300          RETLW   H'00'
                   0301
                   0302 ;******   ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
                   0303 Do_I1State
00A7 0238          0304          MOVF    IState1         ;if IState1 == 0
00A8 0643          0305          BTFSC   STATUS,Z        ;           then Do Start Bit
00A9 0ADD          0306          GOTO    I1StateS
00AA 0201          0307          MOVF    RTCC,W          ;Get current time
00AB 0027          0308          MOVWF   temp
00AC 0219          0309          MOVF    First_RTCC_I1,W ;Get elapsed time; Time Unit = 2 uS
00AD 00A7          0310          SUBWF   temp
00AE 021A          0311          MOVF    nbti1,W         ;Past time for next input bit ?
00AF 0087          0312          SUBWF   temp,W
00B0 0703          0313          BTFSS   STATUS,C
00B1 0AC1          0314          GOTO    _0033
00B2 0218          0315          MOVF    IState1,W       ;Get (0-B) mode #
00B3 0E0F          0316          ANDLW   H'0F'           ;Get only mode #
00B4 01E2          0317          ADDWF   PC              ;jump to subroutine
00B5 0ADD          0318          GOTO    I1StateS        ;Serial Start Bit
00B6 0AE6          0319          GOTO    I1State2        ;1/2 of Start Bit - see if False Start
00B7 0AEF          0320          GOTO    I1State0_7      ;Bit 0
00B8 0AEF          0321          GOTO    I1State0_7      ;Bit 1
00B9 0AEF          0322          GOTO    I1State0_7      ;Bit 2
00BA 0AEF          0323          GOTO    I1State0_7      ;Bit 3
00BB 0AEF          0324          GOTO    I1State0_7      ;Bit 4
00BC 0AEF          0325          GOTO    I1State0_7      ;Bit 5
00BD 0AEF          0326          GOTO    I1State0_7      ;Bit 6
00BE 0AEF          0327          GOTO    I1State0_7      ;Bit 7
00BF 0AF8          0328          GOTO    I1StateE        ;Serial Stop Bit
00C0 0B03          0329          GOTO    I1StateL        ;Last State - End of Stop Bit
                   0330 _0033
00C1 0800          0331          RETLW   H'00'
                   0332
                   0333 ;******   ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
                   0334 Do_I2State
```

© 1994 Microchip Technology Inc.

```
00C2 023C          0335          MOVF    IState2         ;if IState1 == 0
00C3 0643          0336          BTFSC   STATUS,Z        ;              then Do Start Bit
00C4 0B10          0337          GOTO    I2StateS
00C5 0201          0338          MOVF    RTCC,W          ;Get current time
00C6 0027          0339          MOVWF   temp
00C7 021D          0340          MOVF    First_RTCC_I2,W ;Get elapsed time; Time Unit = 2 uS
00C8 00A7          0341          SUBWF   temp
00C9 021E          0342          MOVF    nbti2,W         ;Past time for next input bit ?
00CA 0087          0343          SUBWF   temp,W
00CB 0703          0344          BTFSS   STATUS,C
00CC 0ADC          0345          GOTO    _0035
00CD 021C          0346          MOVF    IState2,W       ;Get (0-B) mode #
00CE 0E0F          0347          ANDLW   H'0F'           ;Get only mode #
00CF 01E2          0348          ADDWF   PC              ;jump to subroutine
00D0 0B10          0349          GOTO    I2StateS        ;Serial Start Bit
00D1 0B19          0350          GOTO    I2StateS2       ;1/2 of Start Bit - see if False Start
00D2 0B22          0351          GOTO    I2State0_7      ;Bit 0
00D3 0B22          0352          GOTO    I2State0_7      ;Bit 1
00D4 0B22          0353          GOTO    I2State0_7      ;Bit 2
00D5 0B22          0354          GOTO    I2State0_7      ;Bit 3
00D6 0B22          0355          GOTO    I2State0_7      ;Bit 4
00D7 0B22          0356          GOTO    I2State0_7      ;Bit 5
00D8 0B22          0357          GOTO    I2State0_7      ;Bit 6
00D9 0B22          0358          GOTO    I2State0_7      ;Bit 7
00DA 0B2B          0359          GOTO    I2StateE        ;Serial Stop Bit
00DB 0B36          0360          GOTO    I2StateL        ;Last State - End of Stop Bit
00DC 0800          0361 _0035    RETLW   H'00'
                   0362
                   0363 ;***               ;Subroutines for Task #2
                   0364 I1StateS           ;Start Bit - Setup timing variables
00DD 0528          0365          BSF     IState1_B       ;Serial Input Active
00DE 0201          0366          MOVF    RTCC,W          ;Store starting time
00DF 0039          0367          MOVWF   First_RTCC_I1
00E0 0C0D          0368          MOVLW   H'0D'           ;Fudge again
00E1 00B9          0369          SUBWF   First_RTCC_I1
00E2 0C32          0370          MOVLW   H'32'           ;Time delay = 1/2 bit time
00E3 003A          0371          MOVWF   nbti1
00E4 02B8          0372          INCF    IState1         ;Increment to next state
00E5 0800          0373          RETLW   H'00'
                   0374 I1State2           ;Check if still a Start Bit
00E6 0705          0375          BTFSS   Serial_IN_1     ;False Start Error ?
00E7 0B06          0376          GOTO    FS_Error_1
00E8 0409          0377          BCF     FS_Flag_1       ;Start Bit OK
00E9 021A          0378          MOVF    nbti1,W         ;Adjust out the error
00EA 01F9          0379          ADDWF   First_RTCC_I1
00EB 0C64          0380          MOVLW   IN_BIT_TIME     ;Time Delay = full bit time
00EC 003A          0381          MOVWF   nbti1
00ED 02B8          0382          INCF    IState1         ;increment to next state
```


© 1994 Microchip Technology Inc.

```
00EE 0800          0383         RETLW   H'00'
                   0384 I1State0_7                  ;Bit 0 - 7
00EF 0705          0385         BTFSS   Serial_IN_1 ;Move Input bit into C
00F0 0403          0386         BCF     STATUS,C
00F1 0605          0387         BTFSC   Serial_IN_1
00F2 0503          0388         BSF     STATUS,C
00F3 033B          0389         RRF     rcv_byte_1  ;Move C into left most bit
00F4 021A          0390         MOVF    nbti1,W
00F5 01F9          0391         ADDWF   First_RTCC_I1 ;Adjust out the error
00F6 02B8          0392         INCF    IState1     ;increment to next state
00F7 0800          0393         RETLW   H'00'
                   0394 I1StateE                    ;Check if we have a proper Stop Bit
00F8 0605          0395         BTFSC   Serial_IN_1 ;Frame Error
00F9 0B09          0396         GOTO    F_Error_1
00FA 0429          0397         BCF     FE_Flag_1   ;Stop Bit OK
00FB 006E          0398         CLRF    T_5_S_CO    ;Reset 5 Sec Timer - got a good byte
                   0399         ;Process the msg Here !
00FC 021B          0400         MOVF    rcv_byte_1,W ;Make a copy of just received byte
00FD 0035          0401         MOVWF   RCV_Storage
00FE 07A8          0402         BTFSS   RCV_Got_One_B ;Report Lost data
00FF 0489          0403         BCF     RCV_Overflow
0100 06A8          0404         BTFSC   RCV_Got_One_B
0101 0589          0405         BSF     RCV_Overflow
0102 05A8          0406         BSF     RCV_Got_One_B ;We Now have a RB Value to go out
                   0407 I1StateL
0103 0078          0408         CLRF    IState1     ;Ready to receive next byte
0104 0428          0409         BCF     IState1_B   ;Serial In not currently active
0105 0800          0410         RETLW   H'00'
                   0411 FS_Error_1                  ;False Start - Shut Down Checking
0106 0428          0412         BCF     IState1_B   ;Serial Input NOT Active
0107 0509          0413         BSF     FS_Flag_1   ;False Start Error
0108 0B03          0414         GOTO    I1StateL    ;Start All Over
                   0415 F_Error_1                   ;Frame Error - Wait for End of Stop Bit
0109 021A          0416         MOVF    nbti1,W     ;Adjust out the error
010A 01F9          0417         ADDWF   First_RTCC_I1
010B 0C32          0418         MOVLW   H'32'       ;Time Delay = 1/2 bit time
010C 003A          0419         MOVWF   nbti1
010D 0529          0420         BSF     FE_Flag_1   ;Frame Error for this Byte ?
010E 02B8          0421         INCF    IState1     ;Increment to next state
010F 0800          0422         RETLW   H'00'
                   0423
                   0424 ;***                ;Subroutines for Task #3
                   0425 I2StateS                    ;Start Bit - Setup timing variables
0110 0548          0426         BSF     IState2_B   ;Serial Input Active
0111 0201          0427         MOVF    RTCC,W      ;Store starting time
0112 003D          0428         MOVWF   First_RTCC_I2
0113 0C0D          0429         MOVLW   H'0D'       ;Fudge again
0114 00BD          0430         SUBWF   First_RTCC_I2
```


© 1994 Microchip Technology Inc.

```
0115 0C32          0431          MOVLW   H'32'          ;Time delay = 1/2 bit time
0116 003E          0432          MOVWF   nbti2
0117 02BC          0433          INCF    IState2       ;Increment to next state
0118 0800          0434          RETLW   H'00'
                   0435 I2StateS2                      ;Check if still a Start Bit
0119 0765          0436          BTFSS   Serial_IN_2   ;False Start Error ?
011A 0B39          0437          GOTO    FS_Error_2
011B 0449          0438          BCF     FS_Flag_2     ;Start Bit OK
011C 021E          0439          MOVF    nbti2,W       ;Adjust out the error
011D 01FD          0440          ADDWF   First_RTCC_I2
011E 0C64          0441          MOVLW   IN_BIT_TIME   ;Time Delay = full bit time
011F 003E          0442          MOVWF   nbti2
0120 02BC          0443          INCF    IState2       ;increment to next state
0121 0800          0444          RETLW   H'00'
                   0445 I2State0_7                     ;Bit 0 - 7
0122 0765          0446          BTFSS   Serial_IN_2   ;Move Input bit into C
0123 0403          0447          BCF     STATUS,C
0124 0665          0448          BTFSC   Serial_IN_2
0125 0503          0449          BSF     STATUS,C
0126 033F          0450          RRF     rcv_byte_2    ;Move C into left most bit
0127 021E          0451          MOVF    nbti2,W
0128 01FD          0452          ADDWF   First_RTCC_I2 ;Adjust out the error
0129 02BC          0453          INCF    IState2       ;increment to next state
012A 0800          0454          RETLW   H'00'
                   0455 I2StateE                       ;Check if we have a proper Stop Bit
012B 0665          0456          BTFSC   Serial_IN_2   ;Frame Error
012C 0B3C          0457          GOTO    F_Error_2
012D 0469          0458          BCF     FE_Flag_2     ;Stop Bit OK
012E 006E          0459          CLRF    T_5_S_CO      ;Reset 5 Sec Timer - got a good byte
                   0460          ;Process the msg Here !
012F 021F          0461          MOVF    rcv_byte_2,W  ;Make a copy of just received byte
0130 0035          0462          MOVWF   RCV_Storage
0131 07A8          0463          BTFSS   RCV_Got_One_B ;Report Lost data
0132 0489          0464          BCF     RCV_Overflow
0133 06A8          0465          BTFSC   RCV_Got_One_B
0134 0589          0466          BSF     RCV_Overflow
0135 05A8          0467          BSF     RCV_Got_One_B ;We Now have a RB Value to go out
                   0468 I2StateL
0136 007C          0469          CLRF    IState2       ;Ready to receive next byte
0137 0448          0470          BCF     IState2_B     ;Serial In not currently active
0138 0800          0471          RETLW   H'00'
                   0472 FS_Error_2
0139 0448          0473          BCF     IState2_B     ;False Start - Shut Down Checking
013A 0549          0474          BSF     FS_Flag_2     ;False Start Error
013B 0B36          0475          GOTO    I2StateL      ;Start All Over
                   0476 F_Error_2                      ;Frame Error - Wait for End of Stop Bit
013C 021E          0477          MOVF    nbti2,W       ;Adjust out the error
013D 01FD          0478          ADDWF   First_RTCC_I2
```


© 1994 Microchip Technology Inc.

```
013E 0C32          0479          MOVLW   H'32'          ;Time Delay = 1/2 bit time
013F 003E          0480          MOVWF   nbti2
0140 0569          0481          BSF     FE_Flag_2     ;Frame Error for this Byte ?
0141 02BC          0482          INCF    IState2       ;Increment to next state
0142 0800          0483          RETLW   H'00'
                   0484
                   0485 ;******          ;Code Starting point
                   0486 Main
0143 0C00          0487          MOVLW   H'00'         ;What is High/Low for RA at INIT State
0144 0025          0488          MOVWF   Port_A
0145 0C00          0489          MOVLW   H'00'         ;What is High/Low for RB at INIT State
0146 0026          0490          MOVWF   Port_B
0147 0CF9          0491          MOVLW   H'F9'         ;RA TRIS at INIT State
0148 0005          0492          TRIS    5
0149 0CFF          0493          MOVLW   H'FF'         ;RB TRIS at INIT State
014A 0006          0494          TRIS    6
014B 0C00          0495          MOVLW   H'00'         ;RTCC/2
014C 0002          0496          OPTION
014D 0900          0497          CALL    Clear_Regs    ;Clear Registers 7-1F - Same Memory Page
014E 0061          0498          CLRF    RTCC          ;Start timers
                   0499
                   0500 ;Initialize Tasks
                   0501                                ;Task #1 waits for byte to output
                   0502                                ;Task #2 waits for Serial IN Start Bit
                   0503                                ;Task #3 waits for Serial IN Start Bit
                   0504                                ;Task #4 runs when Task 1 is Not
                   0505                                ;Task #5 is always running
014F 0206          0506          MOVF    Port_B,W      ;Task #6 is Initialized here
0150 0036          0507          MOVWF   Old_RB
0151 0216          0508          MOVF    Old_RB,W      ;Make all the same initial value
0152 0037          0509          MOVWF   Last_RB
0153 05C8          0510          BSF     RB_NEW_B      ;Tell Task #4: RB byte ready to output
0154 0C08          0511          MOVLW   LED_OFF_MODE
0155 002A          0512          MOVWF   LED_Mode      ;Task #7 is Started
0156 0568          0513          BSF     T_5_S_B       ;Task #8 is Started here
0157 0588          0514          BSF     T_5_M_B       ;Task #9 is Started here
                   0515
                   0516 ; Handle Task & Timer activities - Main Loop
                   0517 Task_1           ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0158 0708          0518          BTFSS   OState_B      ;if not outputing now then skip call
0159 0B5B          0519          GOTO    Task_2
015A 0902          0520          CALL    Do_OState     ;Go Do Task #1
                   0521
                   0522 Task_2           ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
015B 0628          0523          BTFSC   IState1_B     ;if already started then call
015C 0B60          0524          GOTO    _0053
015D 0605          0525          BTFSC   Serial_IN_1   ;if Start bit ? then call
015E 0B60          0526          GOTO    _0053
```

© 1994 Microchip Technology Inc.

```
015F 0B61           0527          GOTO    Task_3
0160 09A7           0528 _0053    CALL    Do_I1State      ;Go Do Task #2
                    0529
                    0530 Task_3           ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
0161 0648           0531          BTFSC   IState2_B       ;if already started then call
0162 0B66           0532          GOTO    _0055
0163 0665           0533          BTFSC   Serial_IN_2     ;if Start bit ? then call
0164 0B66           0534          GOTO    _0055
0165 0B67           0535          GOTO    Task_4
0166 09C2           0536 _0055    CALL    Do_I2State      ;Go Do Task #3
                    0537
                    0538 Task_4           ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0167 0608           0539          BTFSC   OState_B        ;if outputing now then skip call
0168 0B7D           0540          GOTO    _0059
0169 07A8           0541          BTFSS   RCV_Got_One_B   ;Got a NEW Received byte to send
016A 0B70           0542          GOTO    _0057
016B 0215           0543          MOVF    RCV_Storage,W   ;Send just received byte
016C 0033           0544          MOVWF   xmt_byte
016D 04A8           0545          BCF     RCV_Got_One_B   ;Clear need to send old byte
016E 0508           0546          BSF     OState_B        ;Start Task #1 & Lock Out Others
016F 0B7D           0547          GOTO    _0059
0170 07C8           0548 _0057    BTFSS   RB_NEW_B        ;Indicates a change in RB input
0171 0B77           0549          GOTO    _0058
0172 0216           0550          MOVF    Old_RB,W        ;Send New RB value
0173 0033           0551          MOVWF   xmt_byte
0174 04C8           0552          BCF     RB_NEW_B        ;Clear need to send out newest value
0175 0508           0553          BSF     OState_B        ;Start Task #1 & Lock Out Others
0176 0B7D           0554          GOTO    _0059
0177 07E8           0555 _0058    BTFSS   S_5_S_B         ;Serial In 5 secs of inactivity
0178 0B7D           0556          GOTO    _0059
0179 0CFF           0557          MOVLW   H'FF'           ;Tell of inactivity of Serial In
017A 0033           0558          MOVWF   xmt_byte
017B 04E8           0559          BCF     S_5_S_B         ;Clear need to send msg
017C 0508           0560          BSF     OState_B        ;Start Task #1 & Lock Out Others
                    0561
                    0562                  ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
017D 0201           0563 _0059    MOVF    RTCC,W          ;Step-up time units * 512
017E 0027           0564          MOVWF   temp
017F 0211           0565          MOVF    Last_RTCC,W     ;Test to see if it overflowed
0180 0087           0566          SUBWF   temp,W
0181 0703           0567          BTFSS   STATUS,C
0182 0B86           0568          GOTO    Inc_Time
0183 0207           0569          MOVF    temp,W          ;unit error = < |+-512 uS|
0184 0031           0570          MOVWF   Last_RTCC
0185 0B58           0571          GOTO    Task_1
                    0572 Inc_Time
0186 0207           0573          MOVF    temp,W          ;Save current RTCC into Last_RTCC
0187 0031           0574          MOVWF   Last_RTCC
```


© 1994 Microchip Technology Inc.

```
                    0575
                    0576 Task_5         ;Task #5 - Monitor Level Reset Input Line - Always Running !
0188 0606           0577         BTFSC  Level_Reset
0189 0B8C           0578         GOTO   Task_6
018A 0C08           0579         MOVLW  LED_OFF_MODE    ;Lowest Level Indicator output
018B 002A           0580         MOVWF  LED_Mode
                    0581
                    0582 Task_6         ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
018C 02AF           0583         INCF   T_20_mS_CO      ;Inc Counter - Time Unit = 512 uS
018D 0C27           0584         MOVLW  H'27'           ;Used to debounce the input
018E 008F           0585         SUBWF  T_20_mS_CO,W
018F 0743           0586         BTFSS  STATUS,Z
0190 0BA7           0587         GOTO   _0065
0191 006F           0588         CLRF   T_20_mS_CO      ;Reset T_20_mS_CO to start over again
                    0589
0192 0997           0590         CALL   QCheck_T123     ;Quick Check of Tasks #1, #2 and #3
                    0591
0193 0206           0592         MOVF   Port_B,W        ;Last copy of RB same as Current ?
0194 0097           0593         SUBWF  Last_RB,W
0195 0643           0594         BTFSC  STATUS,Z
0196 0B9A           0595         GOTO   _0062
0197 0206           0596         MOVF   Port_B,W        ;  Store Current RB - diff from Last
0198 0037           0597         MOVWF  Last_RB
0199 0B9C           0598         GOTO   _0063
019A 0217           0599 _0062   MOVF   Last_RB,W       ;New Old RB <- same value over 20 mS
019B 0036           0600         MOVWF  Old_RB
019C 0236           0601 _0063   MOVF   Old_RB          ;See if RB is now 0
019D 0643           0602         BTFSC  STATUS,Z        ;RB == 0 ? then keep timer running
019E 0BA1           0603         GOTO   _0064
019F 006C           0604         CLRF   T_5_M_LO        ;Reset 5 Min Timer
01A0 006D           0605         CLRF   T_5_M_HI        ;  still not zero yet
01A1 0901           0606 _0064   CALL   D_H_E_L         ;Determine the Highest Error Level
01A2 07C8           0607         BTFSS  RB_NEW_B        ;Check for Lost Data Error
01A3 04A9           0608         BCF    RB_Overflow
01A4 06C8           0609         BTFSC  RB_NEW_B
01A5 05A9           0610         BSF    RB_Overflow
01A6 05C8           0611         BSF    RB_NEW_B        ;Every 20 mS send Old_RB out
                    0612
                    0613         ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
01A7 0CF9           0614 _0065   MOVLW  H'F9'           ;RA TRIS - refresh
01A8 0005           0615         TRIS   5
01A9 0CFF           0616         MOVLW  H'FF'           ;RB TRIS - refresh
01AA 0006           0617         TRIS   6
01AB 02F4           0618         DECFSZ cc              ;Step-up time units * 256
01AC 0B58           0619         GOTO   Task_1
                    0620
                    0621 Task_7         ;Task 7 - Output Highest Level Indication on LED
01AD 076A           0622         BTFSS  LED_B           ;Is LED active ?
```

© 1994 Microchip Technology Inc.

```
01AE 0BB1          0623         GOTO    Task_8
                   0624
01AF 0997          0625         CALL    QCheck_T123      ;Quick Check of Tasks #1, #2 and #3
                   0626
01B0 0939          0627         CALL    Do_LED           ;Handle LED timing
                   0628
                   0629 Task_8          ;Task #8 - 5 Second Serial Input Lack of Activity Timer
01B1 0768          0630         BTFSS   T_5_S_B          ;5 Sec Timer Active ?
01B2 0BC0          0631         GOTO    Task_9
01B3 02AE          0632         INCF    T_5_S_CO         ;Inc Counter - Time Unit = 131072 uS
01B4 0C26          0633         MOVLW   H'26'            ;Check T_5_S_CO if time
01B5 008E          0634         SUBWF   T_5_S_CO,W
01B6 0743          0635         BTFSS   STATUS,Z
01B7 0BC0          0636         GOTO    Task_9
01B8 006E          0637         CLRF    T_5_S_CO         ;Reset T_5_S_CO
01B9 0C8F          0638         MOVLW   LED_ON_MODE      ;Highest Level Indicator output
01BA 002A          0639         MOVWF   LED_Mode
01BB 07E8          0640         BTFSS   S_5_S_B          ;Check if Lost Data Error
01BC 04C9          0641         BCF     S_5_S_Overflow
01BD 06E8          0642         BTFSC   S_5_S_B
01BE 05C9          0643         BSF     S_5_S_Overflow
01BF 05E8          0644         BSF     S_5_S_B           ;Send notice of 5 seconds of inaction
                   0645
                   0646 Task_9          ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
01C0 0788          0647         BTFSS   T_5_M_B          ;5 Min Timer Active ?
01C1 0BD1          0648         GOTO    Task_A
01C2 02AC          0649         INCF    T_5_M_LO         ;Inc LO Counter; Time Unit = 131072 uS
01C3 0643          0650         BTFSC   STATUS,Z         ;See if carry needs to be passed on ?
01C4 02AD          0651         INCF    T_5_M_HI         ;Inc HI Counter; Time Unit = 131072 uS
01C5 0C08          0652         MOVLW   H'08'            ;#2288<  Check T_5_M_HI if time
01C6 008D          0653         SUBWF   T_5_M_HI,W
01C7 0743          0654         BTFSS   STATUS,Z
01C8 0BD1          0655         GOTO    Task_A
01C9 0CF0          0656         MOVLW   H'F0'            ;#2288>  Check T_5_M_LO if time
01CA 008C          0657         SUBWF   T_5_M_LO,W
01CB 0743          0658         BTFSS   STATUS,Z
01CC 0BD1          0659         GOTO    Task_A
01CD 006C          0660         CLRF    T_5_M_LO         ;Reset T_5_M_LO
01CE 006D          0661         CLRF    T_5_M_HI         ;Reset T_5_M_HI
01CF 0C08          0662         MOVLW   LED_OFF_MODE     ;Lowest Level Indicator output
01D0 002A          0663         MOVWF   LED_Mode
                   0664 Task_A
01D1 0B58          0665         GOTO    Task_1           ;Loop Forever
                   0666
                   0667 ;****
                   0668 Do_D_H_E_L      ; Determine the Highest Error Level & Start Task #7
01D2 0C07          0669         MOVLW   H'07'            ;Check top 7 bits
01D3 0027          0670         MOVWF   temp
```


© 1994 Microchip Technology Inc.

```
01D4 0216           0671          MOVF    Old_RB,W        ;Get copy of 7 debounced Sensor Input
01D5 0037           0672          MOVWF   Last_RB
01D6 0377           0673 _0070    RLF     Last_RB         ;Put top bit into C bit
01D7 0603           0674          BTFSC   STATUS,C        ;Check if C bit is set
01D8 0BDE           0675          GOTO    _0072
01D9 02E7           0676          DECFSZ  temp            ;Continue to check lesser bits
01DA 0BD6           0677          GOTO    _0070
01DB 0206           0678 _0071    MOVF    Port_B,W        ;Restore current value of RB
01DC 0037           0679          MOVWF   Last_RB
01DD 0800           0680          RETLW   H'00'
01DE 020A           0681 _0072    MOVF    LED_Mode,W      ;Get current Level Indicator
01DF 0E07           0682          ANDLW   H'07'           ;Get only       "       "
01E0 0037           0683          MOVWF   Last_RB         ;Store into a tempary register
01E1 0207           0684          MOVF    temp,W          ;Check if already at this Level
01E2 0097           0685          SUBWF   Last_RB,W
01E3 0603           0686          BTFSC   STATUS,C
01E4 0BDB           0687          GOTO    _0071
01E5 0C88           0688          MOVLW   H'88'           ;Start to build LED_Mode
01E6 0107           0689          IORWF   temp,W          ;Put new Level Indicator into reg
01E7 002A           0690          MOVWF   LED_Mode        ;Store new LED Mode
01E8 0BDB           0691          GOTO    _0071
                    0692
                    0693 Do_Clear_Regs  ; Clear Registers 7-1Fh
01E9 0C1F           0694          MOVLW   H'1F'           ;First regs to clear
01EA 0024           0695          MOVWF   FSR
01EB 0060           0696 Loop_C   CLRF    INDIR           ;Clear reg
01EC 00E4           0697          DECF    FSR             ;point to next reg to clear
01ED 0CE7           0698          MOVLW   H'E7'           ;Dec temp, jump if not done
01EE 0084           0699          SUBWF   FSR,W
01EF 0603           0700          BTFSC   STATUS,C
01F0 0BEB           0701          GOTO    Loop_C
01F1 0064           0702          CLRF    FSR             ;Lastly clear FSR reg
01F2 0800           0703          RETLW   H'00'
                    0704
                    0705          ORG     H'1FF'          ;RESET to Main
01FF 0B43           0706          GOTO    Main
                    0707
                    0708          END
                    0709
                    0710

Errors   :     0
Warnings :     0
```

© 1994 Microchip Technology Inc.

```
MPASM 01.00.01 Inter.     APP_D.ASM   7-29-1994  11:20:6                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ; 'Remote Alarm64' V1.00 Designed by Myriad Development Co. - Jerry Farmer
                     0002 ;       PIC16C64, 4MHz Crystal, WatchDog Timer OFF, MPASM instruction set
                     0003
                     0004         list    p=16C64,t=ON,c=132
                     0005
                     0006         include "PIC16C64.H"
                     0001 ;************************  PIC16C64 Header    ************************
                     0002 ;Bank 0 - set RP0 = 0
                     0003
0000                 0004 INDF    equ     00h     ;Indirect Address
0001                 0005 TMR0    equ     01h
0002                 0006 PCL     equ     02h
0003                 0007 STATUS  equ     03h
0004                 0008 FSR     equ     04h
0005                 0009 PORTA   equ     05h     ; I/O Port Assignments
0006                 0010 PORTB   equ     06h
0007                 0011 PORTC   equ     07h
0008                 0012 PORTD   equ     08h
0009                 0013 PORTE   equ     09h
000A                 0014 PCLATH  equ     0Ah
000B                 0015 INTCON  equ     0Bh
000C                 0016 PIR1    equ     0Ch
000E                 0017 TMR1L   equ     0Eh
000F                 0018 TMR1H   equ     0Fh
0010                 0019 T1CON   equ     10h
0011                 0020 TMR2    equ     11h
0012                 0021 T2CON   equ     12h
0013                 0022 SSPBUF  equ     13h
0014                 0023 SSPCON  equ     14h
0015                 0024 CCPR1L  equ     15h
0016                 0025 CCPR1H  equ     16h
0017                 0026 CCP1CON equ     17h
                     0027
                     0028 ;BANK 1 - set RP0 = 1
0001                 0029 OPTION  equ     01h
0005                 0030 TRISA   equ     05h
0006                 0031 TRISB   equ     06h
0007                 0032 TRISC   equ     07h
0008                 0033 TRISD   equ     08h
0009                 0034 TRISE   equ     09h
000C                 0035 PIE1    equ     0Ch
000E                 0036 PCON    equ     0Eh
0012                 0037 PR2     equ     12h
```


© 1994 Microchip Technology Inc.

```
0013                    0038 SSPADD   equ     13h
0014                    0039 SSPSTAT  equ     14h
                        0040 ;
                        0041 ;*******************************************************************************
                        0042 ;
0000                    0043 C        equ     0               ;STATUS
0001                    0044 DC       equ     1               ;STATUS
0002                    0045 Z        equ     2               ;STATUS
0003                    0046 PD       equ     3               ;STATUS
0004                    0047 TO       equ     4               ;STATUS
0005                    0048 RP0      equ     5               ;STATUS
0006                    0049 RP1      equ     6               ;STATUS
0007                    0050 IRP      equ     7               ;STATUS
                        0051
0000                    0052 RBIF     equ     0               ;INTCON
0001                    0053 INTF     equ     1               ;INTCON
0002                    0054 RTIF     equ     2               ;INTCON
0003                    0055 RBIE     equ     3               ;INTCON
0004                    0056 INTE     equ     4               ;INTCON
0005                    0057 RTIE     equ     5               ;INTCON
0006                    0058 PEIE     equ     6               ;INTCON
0007                    0059 GIE      equ     7               ;INTCON
                        0060
0000                    0061 TMR1IF   equ     0               ;PIR1
0001                    0062 TMR2IF   equ     1               ;PIR1
0002                    0063 CCP1IF   equ     2               ;PIR1
0003                    0064 SSPIF    equ     3               ;PIR1
0007                    0065 PSPIF    equ     7               ;PIR1
                        0066
0000                    0067 TMR1ON    equ    0               ;T1CON
0001                    0068 TMR1CS    equ    1               ;T1CON
0002                    0069 T1INSYNC  equ    2               ;T1CON
0003                    0070 T1OSCEN   equ    3               ;T1CON
0004                    0071 T1CKPS0   equ    4               ;T1CON
0005                    0072 T1CKPS1   equ    5               ;T1CON
                        0073
0000                    0074 T2CKPS0   equ    0               ;T2CON
0001                    0075 T2CKPS1   equ    1               ;T2CON
0002                    0076 TMR2ON    equ    2               ;T2CON
0003                    0077 TOUTPS0   equ    3               ;T2CON
0004                    0078 TOUTPS1   equ    4               ;T2CON
0005                    0079 TOUTPS2   equ    5               ;T2CON
0006                    0080 TOUTPS3   equ    6               ;T2CON
                        0081
0000                    0082 SSPM0    equ     0               ;SSPCON
0001                    0083 SSPM1    equ     1               ;SSPCON
0002                    0084 SSPM2    equ     2               ;SSPCON
0003                    0085 SSPM3    equ     3               ;SSPCON
```

© 1994 Microchip Technology Inc.

```
0004                    0086 CKP      equ     4               ;SSPCON
0005                    0087 SSPEN    equ     5               ;SSPCON
0006                    0088 SSPOV    equ     6               ;SSPCON
0007                    0089 WCOL     equ     7               ;SSPCON
                        0090
0000                    0091 CCP1M0   equ     0               ;CCP1CON
0001                    0092 CCP1M1   equ     1               ;CCP1CON
0002                    0093 CCP1M2   equ     2               ;CCP1CON
0003                    0094 CCP1M3   equ     3               ;CCP1CON
0004                    0095 CCP1Y    equ     4               ;CCP1CON
0005                    0096 CCP1X    equ     5               ;CCP1CON
                        0097
0000                    0098 TRISE0   equ     0               ;TRISE
0001                    0099 TRISE1   equ     1               ;TRISE
0002                    0100 TRISE2   equ     2               ;TRISE
0004                    0101 PSPMODE  equ     4               ;TRISE
0005                    0102 IBOV     equ     5               ;TRISE
0006                    0103 OBF      equ     6               ;TRISE
0007                    0104 IBF      equ     7               ;TRISE
                        0105
0000                    0106 TMR1IE   equ     0               ;PIE1
0001                    0107 TMR2IE   equ     1               ;PIE1
0002                    0108 CCP1IE   equ     2               ;PIE1
0003                    0109 SSPIE    equ     3               ;PIE1
0007                    0110 PSPIE    equ     7               ;PIE1
                        0111
0001                    0112 POR      equ     1               ;PCON
                        0113
0001                    0114 Same     equ     1
0000                    0115 W        equ     0
                        0116
0000                    0117 LSB      equ     0
0007                    0118 MSB      equ     7
                        0119
0001                    0120 TRUE     equ     1
0001                    0121 YES      equ     1
0000                    0122 FALSE    equ     0
0000                    0123 NO       equ     0
                        0124 ;
                        0125 ;*****************************************************************************
                        0126
                        0127
                        0128
                        0129
                        0130
                        0006
                        0007
                        0008 ; B Register Definitions
```


© 1994 Microchip Technology Inc.

```
0000                    0009 Serial_IN_1     equ     0               ;Serial Input #1 - 8 bits - INT pin
                        0010                                         ;RB.7 - RB.1 == Input from Sensors
00FF                    0011 RB_TRIS         equ     B'11111111'     ;RB TRIS at INIT State == all input
0000                    0012 RB_MASK         equ     B'00000000'     ;What is High/Low for RB at INIT State
                        0013
                        0014 ; A Register Definitions - Programmable Inputs
0000                    0015 Level_Reset     equ     0               ;PORTA.0 - Reset Level Indicator
0001                    0016 LED             equ     1               ;LED Output - Level/State Indicator
0002                    0017 Serial_Out      equ     2               ;Serial Output - 8 bits + passwords
0003                    0018 PWM_Out         equ     3               ;PWM Output - 8 bits ON, 8 bits OFF
                        0019
00F1                    0020 RA_TRIS         equ     B'11110001'     ;RA TRIS at INIT State
0000                    0021 RA_MASK         equ     B'00000000'     ;What is High/Low for RA at INIT State
                        0022
                        0023 ; Register Files
0020                    0024 temp            equ     20h             ;Tempary holding register - PIC16C54/56
0021                    0025 tmp             equ     21h             ;Tempary reg
0022                    0026 Temp_W          equ     22h             ;Interrupt storage of W
0023                    0027 Temp_Stat       equ     23h             ;Interrupt storage of STATUS
0024                    0028 Temp_FSR        equ     24h             ;Interrupt storage of FSR
0025                    0029 T_B             equ     25h             ;Indicates which Timer(s) are Active = 1
0026                    0030 FLAGS           equ     26h             ;Error Flags
0027                    0031 LED_Mode        equ     27h             ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
0028                    0032 OState          equ     28h             ;Serial Out State
0029                    0033 IState1         equ     29h             ;Serial In #1 State
002A                    0034 cc              equ     2Ah             ;256 * TMR0 time
002B                    0035 T_5_M_LO        equ     2Bh             ;5 Min Timer Counter - Low
002C                    0036 T_5_M_HI        equ     2Ch             ;5 Min Timer Counter - High
002D                    0037 T_5_S_CO        equ     2Dh             ;5 Second Timer - lack of Serial Input
002E                    0038 T_20_mS_CO      equ     2Eh             ;20 mS Timer - used for debouncing
002F                    0039 T_PWM_CO        equ     2Fh             ;PWM Counter
0030                    0040 LED_C           equ     30h             ;LED Counter
0031                    0041 xmt_byte        equ     31h             ;Serial xmit byte - destroyed in use
0032                    0042 rcv_byte_1      equ     32h             ;Receive Serial #1 In byte
0033                    0043 RCV_Storage     equ     33h             ;Long term storage of rcv_byte #1
0034                    0044 Old_RB          equ     34h             ;Oldest/Master copy of RB
0035                    0045 Last_RB         equ     35h             ;Last copy of RB
0036                    0046 PWM_ON          equ     36h             ;PWM ON Counter
0037                    0047 PWM_OFF         equ     37h             ;PWM OFF Counter
0038                    0048 PWM_tmp         equ     38h             ;PWM tempary counter
                        0049
                        0050 ; Indicates which Timer(s) are Active = 1 & Flags - T_B
0000                    0051 OState_B        equ     0               ;Serial Out Active Bit
0001                    0052 IState1_B       equ     1               ;Serial IN #1 Active Bit
0002                    0053 T_5_S_B         equ     2               ;5 Second Timer Active Bit
0003                    0054 T_5_M_B         equ     3               ;5 Min Timer Active Bit
0004                    0055 RCV_Got_One_B   equ     4               ;Got a NEW Received byte to send out
0005                    0056 RB_NEW_B        equ     5               ;Indicates a change in RB input
0006                    0057 S_5_S_B         equ     6               ;Serial In 5 secs of inactivity
```

© 1994 Microchip Technology Inc.

```
0007                    0058 T_PWM_B          equ     7               ;PWM Activity Bit
                        0059
                        0060 ; Error Flags - FLAGS
0000                    0061 FS_Flag_1        equ     0               ;Serial #1 IN had a False Start Error
0001                    0062 FE_Flag_1        equ     1               ;Last Serial #1 IN had a Frame Error
0002                    0063 RCV_Overflow     equ     2               ;Lost Serial Input Byte - too Slow
0003                    0064 RB_Overflow      equ     3               ;Lost RB Input Byte - too Slow
0004                    0065 S_5_S_Overflow   equ     4               ;Lost '5S Inactivity' msg - too Slow
0005                    0066 Time_Bit         equ     5               ;Indicate 512 uS has passed
                        0067
                        0068 ;Equates for LED Task #7 - LED_Mode
0003                    0069 LED_B            equ     3               ;LED is active - LED_Mode.3
0007                    0070 LED_NEW_B        equ     7               ;LED has just changed Modes = 1
0008                    0071 LED_OFF_MODE     equ     B'00001000'     ;LED OFF
0089                    0072 LED_SEQ1_MODE    equ     B'10001001'     ;LED Sequence 1: .2s On, 1s Off
008A                    0073 LED_SEQ2_MODE    equ     B'10001010'     ;LED Sequence 2: 3x(.2s), 1s Off
008B                    0074 LED_SEQ3_MODE    equ     B'10001011'     ;LED Sequence 3: 5x(.2s), 1s Off
009C                    0075 LED_SLOW_MODE    equ     B'10011100'     ;LED Slow Pulsing - .3 Hz
009D                    0076 LED_MEDIUM_MODE  equ     B'10011101'     ;LED Medium Pulsing - 1 Hz
009E                    0077 LED_FAST_MODE    equ     B'10011110'     ;LED Fast Pulsing - 3 Hz
008F                    0078 LED_ON_MODE      equ     B'10001111'     ;LED ON Continuously
                        0079
                        0080         ORG      0                       ;Reset Vector
                        0081
0000 28F7               0082         GOTO     Main
                        0083
                        0084         ORG      4                      ;Interrupt Vector
                        0085
0004 29D4               0086         GOTO     Interrupt
                        0087
                        0088 ;******          Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
                        0089 Do_OState
0005 0828               0090         MOVF     OState,W               ;Get (0-A) mode #
0006 390F               0091         ANDLW    H'0F'                  ;Get only mode #
0007 0782               0092         ADDWF    PCL                    ;jump to subroutine
0008 2814               0093         GOTO     OStateS                ;Serial Start Bit
0009 2823               0094         GOTO     OState0_7              ;Bit 0
000A 2823               0095         GOTO     OState0_7              ;Bit 1
000B 2823               0096         GOTO     OState0_7              ;Bit 2
000C 2823               0097         GOTO     OState0_7              ;Bit 3
000D 2823               0098         GOTO     OState0_7              ;Bit 4
000E 2823               0099         GOTO     OState0_7              ;Bit 5
000F 2823               0100         GOTO     OState0_7              ;Bit 6
0010 2823               0101         GOTO     OState0_7              ;Bit 7
0011 2829               0102         GOTO     OStateE                ;Serial Stop Bit
0012 2831               0103         GOTO     OStateL                ;Last State
0013 0008               0104         RETURN
                        0105
```


© 1994 Microchip Technology Inc.

```
                        0106 OStateS
0014 3000               0107          MOVLW   H'00'             ;Post & Pre 1=1 & OFF
0015 0092               0108          MOVWF   T2CON
0016 1683               0109          BSF     STATUS,RP0        ;Point to BANK 1
0017 3053               0110          MOVLW   H'68' - H'15'     ;104uS - 9600 Baud & adjust to Latency
0018 0092               0111          MOVWF   PR2
0019 1283               0112          BCF     STATUS,RP0        ;Point to BANK 0
001A 0191               0113          CLRF    TMR2              ;Init to 0
001B 108C               0114          BCF     PIR1,TMR2IF       ;Clear Timer 2 Flag so as to start fresh
001C 1505               0115          BSF     PORTA,Serial_Out  ;Output 'Serial Start Bit' starting Now
001D 1512               0116          BSF     T2CON,TMR2ON      ;Start Timer 2
001E 0AA8               0117          INCF    OState            ;inc to next state BEFORE allowing interrupts
                        0118
001F 1683               0119          BSF     STATUS,RP0        ;Point to BANK 1
0020 148C               0120          BSF     PIE1,TMR2IE       ;Allow for Timer 2 interrupts
0021 1283               0121          BCF     STATUS,RP0        ;Point to BANK 0
0022 0008               0122          RETURN
                        0123
                        0124 OState0_7                          ;Bit 0 - 7
0023 0CB1               0125          RRF     xmt_byte          ;Move bit into C from right most bit
0024 1C03               0126          BTFSS   STATUS,C          ;Output C bit
0025 1105               0127          BCF     PORTA,Serial_Out  ;
0026 1803               0128          BTFSC   STATUS,C          ;
0027 1505               0129          BSF     PORTA,Serial_Out  ;
0028 282A               0130          GOTO    OS_End
                        0131 OStateE
0029 1105               0132          BCF     PORTA,Serial_Out  ;Serial Stop Bit
002A 108C               0133 OS_End   BCF     PIR1,TMR2IF       ;Clear Timer 2 Flag so as to start fresh
002B 1683               0134          BSF     STATUS,RP0        ;Point to BANK 1
002C 306C               0135          MOVLW   H'68' + H'4'      ;104uS - 9600 Baud & adjust to Latency
002D 0092               0136          MOVWF   PR2
002E 1283               0137          BCF     STATUS,RP0        ;Point to BANK 0
002F 0AA8               0138          INCF    OState            ;increment to next state
0030 0008               0139          RETURN
                        0140 OStateL
0031 1683               0141          BSF     STATUS,RP0        ;Point to BANK 1
0032 108C               0142          BCF     PIE1,TMR2IE       ;Do NOT Allow Timer 2 interrupts
0033 1283               0143          BCF     STATUS,RP0        ;Point to BANK 0
                        0144
0034 108C               0145          BCF     PIR1,TMR2IF       ;Clear Timer 2 Flag so as to start fresh
0035 1112               0146          BCF     T2CON,TMR2ON      ;Stop Timer 2
0036 01A8               0147          CLRF    OState            ;Ready to send next byte out
0037 1025               0148          BCF     T_B,OState_B      ;Serial Out not active
0038 0008               0149          RETURN
                        0150
                        0151 ;******   Task #7 - Output Highest Level Indication on LED
                        0152 Do_LED
0039 1BA7               0153          BTFSC   LED_Mode,LED_NEW_B ;Initialize regs if change in modes
```

© 1994 Microchip Technology Inc.

```
003A 284C          0154          GOTO    LED_NEW
003B 0AB0          0155          INCF    LED_C           ;Inc Counter - Time Unit = 131072 uS
003C 0827          0156          MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 3907          0157          ANDLW   H'07'           ;Get only mode #
003E 0782          0158          ADDWF   PCL             ;jump to subroutine
003F 2848          0159          GOTO    LED_OFF         ;LED OFF
0040 2864          0160          GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 2867          0161          GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 288A          0162          GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 2850          0163          GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 285E          0164          GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 2861          0165          GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 284D          0166          GOTO    LED_ON          ;LED ON Continuously
0047 0008          0167 _0012    RETURN
                   0168 ;----
                   0169 LED_OFF
0048 1085          0170          BCF     PORTA,LED       ;Turn off LED
0049 11A7          0171          BCF     LED_Mode,LED_B  ;LED must be off
004A 01B0          0172          CLRF    LED_C           ;Reset Counter - LED_C = 0
004B 0008          0173          RETURN
                   0174 ;----
                   0175 LED_NEW
004C 13A7          0176          BCF     LED_Mode,LED_NEW_B  ;Done initializing
                   0177 LED_ON
004D 1485          0178          BSF     PORTA,LED       ;Turn on LED
004E 01B0          0179          CLRF    LED_C           ;Reset Counter - LED_C = 0
004F 0008          0180          RETURN
                   0181 ;----
                   0182 LED_SLOW
0050 300C          0183          MOVLW   H'0C'           ;.3Hz @ 50% Duty
0051 00A0          0184          MOVWF   temp
0052 0820          0185 LED_S    MOVF    temp,W          ;Check LED_C if time, .3Hz @ 50% Duty
0053 0230          0186          SUBWF   LED_C,W
0054 1D03          0187          BTFSS   STATUS,Z
0055 2847          0188          GOTO    _0012
0056 3010          0189          MOVLW   H'10'
0057 06A7          0190          XORWF   LED_Mode        ;Switch states
0058 1E27          0191          BTFSS   LED_Mode,4      ;Now make LED same state
0059 1085          0192          BCF     PORTA,LED
005A 1A27          0193          BTFSC   LED_Mode,4
005B 1485          0194          BSF     PORTA,LED
005C 01B0          0195          CLRF    LED_C           ;Reset LED_C
005D 0008          0196          RETURN
                   0197 ;----
                   0198 LED_MEDIUM
005E 3004          0199          MOVLW   H'04'           ;1Hz @ 50% Duty
005F 00A0          0200          MOVWF   temp
0060 2852          0201          GOTO    LED_S           ;Go do it
```


© 1994 Microchip Technology Inc.

```
                0202 ;---
                0203 LED_FAST
0061 3001       0204        MOVLW   H'01'           ;3Hz @ 50% Duty
0062 00A0       0205        MOVWF   temp
0063 2852       0206        GOTO    LED_S           ;Go do it
                0207 ;---
                0208 LED_SEQ1                       ;.2 ON, 1 OFF
0064 1E27       0209        BTFSS   LED_Mode,4      ;Skip if bit is high
0065 2876       0210        GOTO    ON1             ;Go do it
0066 2882       0211        GOTO    OFF3            ;Go do it
                0212 ;---
                0213 LED_SEQ2                       ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 0827       0214        MOVF    LED_Mode,W
0068 00A0       0215        MOVWF   temp
0069 3030       0216        MOVLW   H'30'           ;Get sequence # only
006A 05A0       0217        ANDWF   temp
006B 0EA0       0218        SWAPF   temp            ;swap nibbles
006C 0820       0219        MOVF    temp,W          ;get nibble for offset
006D 0782       0220        ADDWF   PCL             ;Table jump calculation
006E 2876       0221        GOTO    ON1             ;LED is on, check if time to change
006F 287C       0222        GOTO    OFF2            ;LED is off, check if time to change
0070 2876       0223        GOTO    ON1             ;LED is on, check if time to change
0071 2882       0224        GOTO    OFF3            ;LED is off, check if time to change
                0225 ;---
                0226 LED_Exit
0072 3010       0227        MOVLW   H'10'           ;Inc Seq #
0073 07A7       0228        ADDWF   LED_Mode
0074 01B0       0229        CLRF    LED_C           ;Reset LED_C
0075 0008       0230        RETURN
                0231 ON1
0076 3002       0232        MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
0077 0230       0233        SUBWF   LED_C,W
0078 1D03       0234        BTFSS   STATUS,Z
0079 2847       0235        GOTO    _0012
007A 1085       0236        BCF     PORTA,LED       ;Turn off LED
007B 2872       0237        GOTO    LED_Exit
                0238 OFF2
007C 3002       0239        MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0230       0240        SUBWF   LED_C,W
007E 1D03       0241        BTFSS   STATUS,Z
007F 2847       0242        GOTO    _0012
0080 1485       0243        BSF     PORTA,LED       ;Turn on LED
0081 2872       0244        GOTO    LED_Exit
                0245 OFF3
0082 3008       0246        MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0230       0247        SUBWF   LED_C,W
0084 1D03       0248        BTFSS   STATUS,Z
0085 2847       0249        GOTO    _0012
```

© 1994 Microchip Technology Inc.

```
0086 1485          0250          BSF     PORTA,LED       ;Turn on LED
0087 30F0          0251          MOVLW   H'F0'
0088 04A7          0252          IORWF   LED_Mode        ;Cause (Seq# & NEW) to overflow to 0
0089 2872          0253          GOTO    LED_Exit
                   0254 LED_SEQ3                         ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 0827          0255          MOVF    LED_Mode,W      ;Get LED info
008B 00A0          0256          MOVWF   temp
008C 3070          0257          MOVLW   H'70'           ;Get sequence # only
008D 05A0          0258          ANDWF   temp
008E 0EA0          0259          SWAPF   temp            ;swap nibbles
008F 0820          0260          MOVF    temp,W          ;get nibble for offset
0090 0782          0261          ADDWF   PCL             ;Table jump calculation
0091 2876          0262          GOTO    ON1             ;LED is on check if time to change
0092 287C          0263          GOTO    OFF2            ;LED is off check if time to change
0093 2876          0264          GOTO    ON1             ;LED is on check if time to change
0094 287C          0265          GOTO    OFF2            ;LED is off check if time to change
0095 2876          0266          GOTO    ON1             ;LED is on check if time to change
0096 2882          0267          GOTO    OFF3            ;LED is off check if time to change
                   0268
                   0269 ;******           Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
                   0270 Do_I1State
0097 0829          0271          MOVF    IState1,W       ;Get (0-B) mode #
0098 390F          0272          ANDLW   H'0F'           ;Get only mode #
0099 0782          0273          ADDWF   PCL             ;jump to subroutine
009A 28A7          0274          GOTO    I1StateS        ;Serial Start Bit
009B 28B8          0275          GOTO    I1State2        ;1/2 of Start Bit - see if False Start
009C 28C5          0276          GOTO    I1State0_7      ;Bit 0
009D 28C5          0277          GOTO    I1State0_7      ;Bit 1
009E 28C5          0278          GOTO    I1State0_7      ;Bit 2
009F 28C5          0279          GOTO    I1State0_7      ;Bit 3
00A0 28C5          0280          GOTO    I1State0_7      ;Bit 4
00A1 28C5          0281          GOTO    I1State0_7      ;Bit 5
00A2 28C5          0282          GOTO    I1State0_7      ;Bit 6
00A3 28C5          0283          GOTO    I1State0_7      ;Bit 7
00A4 28D4          0284          GOTO    I1StateE        ;Serial Stop Bit
00A5 28DF          0285          GOTO    I1StateL        ;Last State - End of Stop Bit
00A6 0008          0286          RETURN
                   0287
                   0288 ;***              Subroutines for Task #2
                   0289 I1StateS                         ;Start Bit - Setup timing variables
00A7 120B          0290          BCF     INTCON,INTE     ;Disable detecting changes on INT pin
00A8 108B          0291          BCF     INTCON,INTF     ;Clear Interrupting Flag
00A9 3000          0292          MOVLW   H'00'           ;Internal Clk, Pre 1=1 & OFF
00AA 0090          0293          MOVWF   T1CON
00AB 018E          0294          CLRF    TMR1L           ;Calculate (0 - #) of counts until roll-over
00AC 304A          0295          MOVLW   H'68' - H'1E'   ;208us/2 = 4800 Baud & adjust to Latency
00AD 028E          0296          SUBWF   TMR1L
00AE 018F          0297          CLRF    TMR1H
```


© 1994 Microchip Technology Inc.

```
00AF 038F          0298          DECF    TMR1H           ;H'FF'
00B0 100C          0299          BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00B1 1410          0300          BSF     T1CON,TMR1ON    ;Start Timer 1
00B2 0AA9          0301          INCF    IState1         ;inc to next state BEFORE allowing interrupts
00B3 14A5          0302          BSF     T_B,IState1_B   ;Serial Input Active
                   0303
00B4 1683          0304          BSF     STATUS,RP0      ;Point to BANK 1
00B5 140C          0305          BSF     PIE1,TMR1IE     ;Allow for Timer 1 interrupts
00B6 1283          0306          BCF     STATUS,RP0      ;Point to BANK 0
00B7 0008          0307          RETURN
                   0308 I1State2                        ;Check if still a Start Bit
00B8 1C06          0309          BTFSS   PORTB,Serial_IN_1  ;False Start Error ?
00B9 28E9          0310          GOTO    FS_Error_1
00BA 1026          0311          BCF     FLAGS,FS_Flag_1 ;Start Bit OK
00BB 1010          0312          BCF     T1CON,TMR1ON    ;Stop Timer 1
00BC 018E          0313          CLRF    TMR1L
00BD 30AA          0314          MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00BE 028E          0315          SUBWF   TMR1L
00BF 018F          0316          CLRF    TMR1H
00C0 038F          0317          DECF    TMR1H           ;H'FF'
00C1 100C          0318          BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00C2 1410          0319          BSF     T1CON,TMR1ON    ;Start Timer 1
00C3 0AA9          0320          INCF    IState1         ;increment to next state
00C4 0008          0321          RETURN
                   0322 I1State0_7                      ;Bit 0 - 7
00C5 1C06          0323          BTFSS   PORTB,Serial_IN_1  ;Move Input bit into C
00C6 1003          0324          BCF     STATUS,C
00C7 1806          0325          BTFSC   PORTB,Serial_IN_1
00C8 1403          0326          BSF     STATUS,C
00C9 0CB2          0327          RRF     rcv_byte_1      ;Move C into left most bit
00CA 1010          0328          BCF     T1CON,TMR1ON    ;Stop Timer 1
00CB 018E          0329          CLRF    TMR1L
00CC 30AA          0330          MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00CD 028E          0331          SUBWF   TMR1L
00CE 018F          0332          CLRF    TMR1H
00CF 038F          0333          DECF    TMR1H           ;H'FF'
00D0 100C          0334          BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00D1 1410          0335          BSF     T1CON,TMR1ON    ;Start Timer 1
00D2 0AA9          0336          INCF    IState1         ;increment to next state
00D3 0008          0337          RETURN
                   0338 I1StateE                        ;Check if we have a proper Stop Bit
00D4 1806          0339          BTFSC   PORTB,Serial_IN_1  ;Frame Error
00D5 28EC          0340          GOTO    F_Error_1
00D6 10A6          0341          BCF     FLAGS,FE_Flag_1 ;Stop Bit OK
00D7 01AD          0342          CLRF    T_5_S_CO        ;Reset 5 Sec Timer - got a good byte
                   0343          ;Process the msg Here !
00D8 0832          0344          MOVF    rcv_byte_1,W    ;Make a copy of just received byte
00D9 00B3          0345          MOVWF   RCV_Storage
```

© 1994 Microchip Technology Inc.

```
00DA 1E25  0346          BTFSS   T_B,RCV_Got_One_B  ;Report Lost data
00DB 1126  0347          BCF     FLAGS,RCV_Overflow
00DC 1A25  0348          BTFSC   T_B,RCV_Got_One_B
00DD 1526  0349          BSF     FLAGS,RCV_Overflow
00DE 1625  0350          BSF     T_B,RCV_Got_One_B  ;We Now have a RB Value to go out
           0351 I1StateL
00DF 1010  0352          BCF     T1CON,TMR1ON    ;Stop Timer 1
00E0 1683  0353          BSF     STATUS,RP0      ;Point to BANK 1
00E1 140C  0354          BSF     PIE1,TMR1IE     ;Allow for Timer 1 interrupts
00E2 1283  0355          BCF     STATUS,RP0      ;Point to BANK 0
00E3 100C  0356          BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
           0357
00E4 01A9  0358          CLRF    IState1         ;Ready to receive next byte
00E5 10A5  0359          BCF     T_B,IState1_B   ;Serial In not currently active
           0360
00E6 108B  0361          BCF     INTCON,INTF     ;Clear Interrupting Flag
00E7 160B  0362          BSF     INTCON,INTE     ;Enable detecting changes on INT pin
00E8 0008  0363          RETURN
           0364 FS_Error_1                      ;False Start - Shut Down Checking
00E9 10A5  0365          BCF     T_B,IState1_B   ;Serial Input NOT Active
00EA 1426  0366          BSF     FLAGS,FS_Flag_1  ;False Start Error
00EB 28DF  0367          GOTO    I1StateL        ;Start All Over
           0368 F_Error_1                       ;Frame Error - Wait for End of Stop Bit
00EC 14A6  0369          BSF     FLAGS,FE_Flag_1 ;Frame Error for this Byte ?
00ED 0AA9  0370          INCF    IState1         ;Increment to next state
00EE 1010  0371          BCF     T1CON,TMR1ON    ;Stop Timer 1
00EF 018E  0372          CLRF    TMR1L
00F0 304A  0373          MOVLW   H'68' - H'1E'   ;208us/2 = 4800 Baud & adjust to Latency
00F1 028E  0374          SUBWF   TMR1L
00F2 018F  0375          CLRF    TMR1H
00F3 038F  0376          DECF    TMR1H           ;H'FF'
00F4 100C  0377          BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00F5 1410  0378          BSF     T1CON,TMR1ON    ;Start Timer 1
00F6 0008  0379          RETURN
           0380
           0381 ;******          Code Starting point
           0382 Main
00F7 0183  0383          CLRF    STATUS
00F8 0184  0384          CLRF    FSR
00F9 0181  0385          CLRF    TMR0            ;Clear Timer0
00FA 3000  0386          MOVLW   H'00'           ;What is High/Low for RA at RESET State
00FB 0085  0387          MOVWF   PORTA
00FC 0086  0388          MOVWF   PORTB
00FD 0087  0389          MOVWF   PORTC
00FE 0088  0390          MOVWF   PORTD
00FF 0089  0391          MOVWF   PORTE
0100 018A  0392          CLRF    PCLATH
0101 3060  0393          MOVLW   H'60'           ;/GIE,PEIE,RTIE,/INTE,/RBIE,/RTIF,/INTF,/RBIF
```


© 1994 Microchip Technology Inc.

```
0102 008B          0394          MOVWF   INTCON
0103 018C          0395          CLRF    PIR1            ;Timer 2 Flag cleared
0104 018E          0396          CLRF    TMR1L
0105 018F          0397          CLRF    TMR1H
0106 0190          0398          CLRF    T1CON           ;Timer 1 OFF until ready for input
0107 0191          0399          CLRF    TMR2
0108 0192          0400          CLRF    T2CON           ;Timer 2 OFF until have byte to output
0109 0193          0401          CLRF    SSPBUF
010A 0194          0402          CLRF    SSPCON
010B 0195          0403          CLRF    CCPR1L
010C 0196          0404          CLRF    CCPR1H
010D 0197          0405          CLRF    CCP1CON
                   0406
010E 1683          0407          BSF     STATUS,RP0      ;Point to BANK 1
010F 3040          0408          MOVLW   H'40'           ;TMR0/2 & Interrupt on Rising edge of INT
0110 0081          0409          MOVWF   OPTION          ;Load OPTION reg
0111 30F1          0410          MOVLW   RA_TRIS
0112 0085          0411          MOVWF   TRISA
0113 30FF          0412          MOVLW   H'FF'           ;RB TRIS at RESET State
0114 0086          0413          MOVWF   TRISB
0115 0087          0414          MOVWF   TRISC
0116 0088          0415          MOVWF   TRISD
0117 3007          0416          MOVLW   H'07'           ;PSPMODE=0
0118 0089          0417          MOVWF   TRISE
0119 018C          0418          CLRF    PIE1            ;Timer 2 Interrupt disabled
011A 148E          0419          BSF     PCON,POR
011B 0192          0420          CLRF    PR2
011C 0193          0421          CLRF    SSPADD
011D 0194          0422          CLRF    SSPSTAT
                   0423
011E 1283          0424          BCF     STATUS,RP0      ;Point to BANK 0
011F 21BF          0425          CALL    Clear_Regs      ;Clear Registers 20-7F, A0-C0 Memory Pages
                   0426
                   0427 ;Initialize Tasks
                   0428                                  ;Task #1 waits for byte to output
                   0429                                  ;Task #2 waits for Serial IN Start Bit
0120 3031          0430          MOVLW   H'31'           ;Task #3 is initialized for square pulses
0121 00B6          0431          MOVWF   PWM_ON          ;   "      25 mS ON
0122 3031          0432          MOVLW   H'31'           ;   "              Period = 50 mS, DS= 50%
0123 00B7          0433          MOVWF   PWM_OFF         ;   "      25 mS OFF
0124 01AF          0434          CLRF    T_PWM_CO        ;   "
0125 0836          0435          MOVF    PWM_ON,W        ;mov PWM_tmp,PWM_ON
0126 00B8          0436          MOVWF   PWM_tmp         ; "
0127 1585          0437          BSF     PORTA,PWM_Out   ;Start Outputing ON
0128 17A5          0438          BSF     T_B,T_PWM_B     ;   "
                   0439                                  ;Task #4 runs when Task 1 is Not
                   0440                                  ;Task #5 is always running
0129 0806          0441          MOVF    PORTB,W         ;Task #6 is Initialized here
```


© 1994 Microchip Technology Inc.

```
012A 00B4          0442          MOVWF   Old_RB
012B 0834          0443          MOVF    Old_RB,W        ;Make all the same initial value
012C 00B5          0444          MOVWF   Last_RB
012D 16A5          0445          BSF     T_B,RB_NEW_B    ;Tell Task #4: RB byte ready to output
012E 3008          0446          MOVLW   LED_OFF_MODE
012F 00A7          0447          MOVWF   LED_Mode        ;Task #7 is Started
0130 1525          0448          BSF     T_B,T_5_S_B     ;Task #8 is Started here
0131 15A5          0449          BSF     T_B,T_5_M_B     ;Task #9 is Started here
                   0450
0132 178B          0451          BSF     INTCON,GIE      ;Enable Global Interrupts
                   0452
                   0453 ; Handle Task & Timer activities - Main Loop done in background
                   0454 Inc_Time         ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
0133 1EA6          0455          BTFSS   FLAGS,Time_Bit  ;Idle Task - wait until 512 uS has gone by
0134 2933          0456          GOTO    Inc_Time
0135 12A6          0457          BCF     FLAGS,Time_Bit  ;Reset for next indicator from TMR0 Interrupt
                   0458
                   0459 Task_3           ;Task #3 - PWM, Period = (PWM_ON + PWM_OFF) * 512uS
0136 1FA5          0460          BTFSS   T_B,T_PWM_B     ;if NOT outputing now then skip call
0137 2947          0461          GOTO    Task_4
0138 0AAF          0462          INCF    T_PWM_CO        ;Inc count of time
0139 0838          0463          MOVF    PWM_tmp,W       ;cjne T_PWM_CO,PWM_tmp,Task_4
013A 022F          0464          SUBWF   T_PWM_CO,W      ; "
013B 1D03          0465          BTFSS   STATUS,Z        ; "
013C 2947          0466          GOTO    Task_4          ; "
013D 01AF          0467          CLRF    T_PWM_CO        ;Reset timer
013E 1985          0468          BTFSC   PORTA,PWM_Out
013F 2944          0469          GOTO    T3_1
0140 1585          0470          BSF     PORTA,PWM_Out   ;Change Output State
0141 0836          0471          MOVF    PWM_ON,W        ;mov PWM_tmp,PWM_ON
0142 00B8          0472          MOVWF   PWM_tmp         ; "
0143 2947          0473          GOTO    Task_4
                   0474 T3_1
0144 1185          0475          BCF     PORTA,PWM_Out   ;Change Output State
0145 0837          0476          MOVF    PWM_OFF,W       ;mov PWM_tmp,PWM_OFF
0146 00B8          0477          MOVWF   PWM_tmp       ; "
                   0478
                   0479 Task_4           ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0147 1825          0480          BTFSC   T_B,OState_B    ;if outputing now then skip call
0148 295E          0481          GOTO    Task_5
0149 1E25          0482          BTFSS   T_B,RCV_Got_One_B  ;Got a NEW Received byte to send
014A 2950          0483          GOTO    _0057
014B 0833          0484          MOVF    RCV_Storage,W   ;Send just received byte
014C 00B1          0485          MOVWF   xmt_byte
014D 1225          0486          BCF     T_B,RCV_Got_One_B  ;Clear need to send old byte
014E 1425          0487          BSF     T_B,OState_B    ;Start Task #1 & Lock Out Others
014F 295D          0488          GOTO    T4_S
0150 1EA5          0489 _0057    BTFSS   T_B,RB_NEW_B    ;Indicates a change in RB input
```


© 1994 Microchip Technology Inc.

```
0151 2957          0490          GOTO    _0058
0152 0834          0491          MOVF    Old_RB,W        ;Send New RB value
0153 00B1          0492          MOVWF   xmt_byte
0154 12A5          0493          BCF     T_B,RB_NEW_B    ;Clear need to send out newest value
0155 1425          0494          BSF     T_B,OState_B    ;Start Task #1 & Lock Out Others
0156 295D          0495          GOTO    T4_S
0157 1F25          0496 _0058    BTFSS   T_B,S_5_S_B     ;Serial In 5 secs of inactivity
0158 295E          0497          GOTO    Task_5
0159 30FF          0498          MOVLW   H'FF'           ;Tell of inactivity of Serial In
015A 00B1          0499          MOVWF   xmt_byte
015B 1325          0500          BCF     T_B,S_5_S_B     ;Clear need to send msg
015C 1425          0501          BSF     T_B,OState_B    ;Start Task #1 & Lock Out Others
                   0502 T4_S                             ;Start Task #1
015D 2005          0503          CALL    Do_OState
                   0504
                   0505 Task_5                ;Task #5 - Monitor Level Reset Input Line - Always Running !
015E 1805          0506          BTFSC   PORTA,Level_Reset
015F 2962          0507          GOTO    Task_6
0160 3008          0508          MOVLW   LED_OFF_MODE    ;Lowest Level Indicator output
0161 00A7          0509          MOVWF   LED_Mode
                   0510
                   0511 Task_6                ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
0162 0AAE          0512          INCF    T_20_mS_CO      ;Inc Counter - Time Unit = 512 uS
0163 3027          0513          MOVLW   H'27'           ;Used to debounce the input
0164 022E          0514          SUBWF   T_20_mS_CO,W
0165 1D03          0515          BTFSS   STATUS,Z
0166 297C          0516          GOTO    _0065
0167 01AE          0517          CLRF    T_20_mS_CO      ;Reset T_20_mS_CO to start over again
0168 0806          0518          MOVF    PORTB,W         ;Last copy of RB same as Current ?
0169 0235          0519          SUBWF   Last_RB,W
016A 1903          0520          BTFSC   STATUS,Z
016B 296F          0521          GOTO    _0062
016C 0806          0522          MOVF    PORTB,W         ;Store Current RB - diff from Last
016D 00B5          0523          MOVWF   Last_RB
016E 2971          0524          GOTO    _0063
016F 0835          0525 _0062    MOVF    Last_RB,W       ;New Old RB <- same value over 20 mS
0170 00B4          0526          MOVWF   Old_RB
0171 08B4          0527 _0063    MOVF    Old_RB          ;See if RB is now 0
0172 1903          0528          BTFSC   STATUS,Z        ;RB == 0 ? then keep timer running
0173 2976          0529          GOTO    _0064
0174 01AB          0530          CLRF    T_5_M_LO        ;Reset 5 Min Timer
0175 01AC          0531          CLRF    T_5_M_HI        ;  still not zero yet
0176 21A8          0532 _0064    CALL    D_H_E_L         ;Determine the Highest Error Level
0177 1EA5          0533          BTFSS   T_B,RB_NEW_B    ;Check for Lost Data Error
0178 11A6          0534          BCF     FLAGS,RB_Overflow
0179 1AA5          0535          BTFSC   T_B,RB_NEW_B
017A 15A6          0536          BSF     FLAGS,RB_Overflow
017B 16A5          0537          BSF     T_B,RB_NEW_B    ;Every 20 mS send Old_RB out
```

© 1994 Microchip Technology Inc.

```
                        0538
                        0539            ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
                        0540 _0065
017C 1683               0541            BSF     STATUS,RP0      ;Point to BANK 1
017D 30F1               0542            MOVLW   RA_TRIS         ;RA TRIS - refresh
017E 0085               0543            MOVWF   TRISA
017F 30FF               0544            MOVLW   H'FF'           ;RB TRIS - refresh
0180 0086               0545            MOVWF   TRISB
0181 1283               0546            BCF     STATUS,RP0      ;Point to BANK 0
0182 0BAA               0547            DECFSZ  cc              ;Step-up time units * 256
0183 2933               0548            GOTO    Inc_Time
                        0549
                        0550 Task_7     ;Task 7 - Output Highest Level Indication on LED
0184 1DA7               0551            BTFSS   LED_Mode,LED_B  ;Is LED active ?
0185 2987               0552            GOTO    Task_8
0186 2039               0553            CALL    Do_LED          ;Handle LED timing
                        0554
                        0555 Task_8     ;Task #8 - 5 Second Serial Input Lack of Activity Timer
0187 1D25               0556            BTFSS   T_B,T_5_S_B     ;5 Sec Timer Active ?
0188 2996               0557            GOTO    Task_9
0189 0AAD               0558            INCF    T_5_S_CO        ;Inc Counter - Time Unit = 131072 uS
018A 3026               0559            MOVLW   H'26'           ;Check T_5_S_CO if time
018B 022D               0560            SUBWF   T_5_S_CO,W
018C 1D03               0561            BTFSS   STATUS,Z
018D 2996               0562            GOTO    Task_9
018E 01AD               0563            CLRF    T_5_S_CO        ;Reset T_5_S_CO
018F 308F               0564            MOVLW   LED_ON_MODE     ;Highest Level Indicator output
0190 00A7               0565            MOVWF   LED_Mode
0191 1F25               0566            BTFSS   T_B,S_5_S_B     ;Check if Lost Data Error
0192 1226               0567            BCF     FLAGS,S_5_S_Overflow
0193 1B25               0568            BTFSC   T_B,S_5_S_B
0194 1626               0569            BSF     FLAGS,S_5_S_Overflow
0195 1725               0570            BSF     T_B,S_5_S_B     ;Send notice of 5 seconds of inaction
                        0571
                        0572 Task_9     ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
0196 1DA5               0573            BTFSS   T_B,T_5_M_B     ;5 Min Timer Active ?
0197 29A7               0574            GOTO    Task_A
0198 0AAB               0575            INCF    T_5_M_LO        ;Inc LO Counter; Time Unit = 131072 uS
0199 1903               0576            BTFSC   STATUS,Z        ;See if carry needs to be passed on ?
019A 0AAC               0577            INCF    T_5_M_HI        ;Inc HI Counter; Time Unit = 131072 uS
019B 3008               0578            MOVLW   H'08'           ;#2288<  Check T_5_M_HI if time
019C 022C               0579            SUBWF   T_5_M_HI,W
019D 1D03               0580            BTFSS   STATUS,Z
019E 29A7               0581            GOTO    Task_A
019F 30F0               0582            MOVLW   H'F0'           ;#2288>  Check T_5_M_LO if time
01A0 022B               0583            SUBWF   T_5_M_LO,W
01A1 1D03               0584            BTFSS   STATUS,Z
01A2 29A7               0585            GOTO    Task_A
```


© 1994 Microchip Technology Inc.

```
01A3 01AB          0586          CLRF    T_5_M_LO        ;Reset T_5_M_LO
01A4 01AC          0587          CLRF    T_5_M_HI        ;Reset T_5_M_HI
01A5 3008          0588          MOVLW   LED_OFF_MODE    ;Lowest Level Indicator output
01A6 00A7          0589          MOVWF   LED_Mode
                   0590 Task_A
01A7 2933          0591          GOTO    Inc_Time        ;Loop Forever
                   0592
                   0593 ;****    Determine the Highest Error Level & Start Task #7
                   0594 D_H_E_L
01A8 3007          0595          MOVLW   H'07'           ;Check top 7 bits
01A9 00A0          0596          MOVWF   temp
01AA 0834          0597          MOVF    old_RB,W        ;Get copy of 7 debounced Sensor Input
01AB 00A1          0598          MOVWF   tmp
01AC 0DA1          0599 _0070    RLF     tmp             ;Put top bit into C bit
01AD 1803          0600          BTFSC   STATUS,C        ;Check if C bit is set
01AE 29B4          0601          GOTO    _0072
01AF 0BA0          0602          DECFSZ  temp            ;Continue to check lesser bits
01B0 29AC          0603          GOTO    _0070
01B1 0806          0604 _0071    MOVF    PORTB,W         ;Restore current value of RB
01B2 00A1          0605          MOVWF   tmp
01B3 0008          0606          RETURN
01B4 0827          0607 _0072    MOVF    LED_Mode,W      ;Get current Level Indicator
01B5 3907          0608          ANDLW   H'07'           ;Get only       "       "
01B6 00A1          0609          MOVWF   tmp             ;Store into a tempary register
01B7 0820          0610          MOVF    temp,W          ;Check if already at this Level
01B8 0221          0611          SUBWF   tmp,W
01B9 1803          0612          BTFSC   STATUS,C
01BA 29B1          0613          GOTO    _0071
01BB 3088          0614          MOVLW   H'88'           ;Start to build LED_Mode
01BC 0420          0615          IORWF   temp,W          ;Put new Level Indicator into reg
01BD 00A7          0616          MOVWF   LED_Mode        ;Store new LED Mode
01BE 29B1          0617          GOTO    _0071
                   0618
                   0619 ;******  Clear Registers 20-7Fh, A0-C0
                   0620 Clear_Regs
01BF 307F          0621 ******   MOVLW   H'7F'           ;First regs to clear in Bank 0
01C0 0084          0622          MOVWF   FSR
01C1 0180          0623 Loop_C1  CLRF    INDF            ;Clear reg
01C2 0384          0624          DECF    FSR             ;point to next reg to clear
01C3 3020          0625          MOVLW   H'20'           ;Dec temp, jump if not done
01C4 0204          0626          SUBWF   FSR,W
01C5 1803          0627 ******   BTFSC   STATUS,C
01C6 29C1          0628          GOTO    Loop_C1
                   0629
01C7 30C0          0630          MOVLW   H'C0'           ;First regs to clear in Bank 1
01C8 0084          0631          MOVWF   FSR
01C9 0180          0632 Loop_C2  CLRF    INDF            ;Clear reg
01CA 0384          0633          DECF    FSR             ;point to next reg to clear
```

© 1994 Microchip Technology Inc.

```
01CB 30A0          0634          MOVLW   H'A0'           ;Dec temp, jump if not done
01CC 0204          0635          SUBWF   FSR,W
01CD 1803          0636          BTFSC   STATUS,C
01CE 29C9          0637          GOTO    Loop_C2
01CF 0184          0638          CLRF    FSR             ;Lastly clear FSR reg
01D0 0008          0639          RETURN
                   0640
                   0641 ;******   TMR0 IRS - Set Time_Bit for background tasks
                   0642 Do_Inc_Time
01D1 16A6          0643          BSF     FLAGS,Time_Bit  ;Tell background tasks of overflow
01D2 110B          0644          BCF     INTCON,RTIF     ;Clear for next overflow
01D3 0008          0645          RETURN
                   0646
                   0647 ;******   Handle Interrupts Here
                   0648 Interrupt
01D4 00A2          0649 PUSH:    MOVWF   Temp_W
01D5 0E03          0650          SWAPF   STATUS,W
01D6 00A3          0651          MOVWF   Temp_Stat
01D7 0804          0652          MOVF    FSR,W
01D8 00A4          0653          MOVWF   Temp_FSR
01D9 1283          0654          BCF     STATUS,RP0      ;Point to BANK 0 - Very IMPORTANT !!!!!
                   0655
01DA 188C          0656          BTFSC   PIR1,TMR2IF
01DB 2005          0657          CALL    Do_OState       ;Go Do Task #1 - all states
                   0658
01DC 18A5          0659          BTFSC   T_B,IState1_B   ;INTF will set even if INTE is cleared
01DD 29E0          0660          GOTO    I1
01DE 188B          0661          BTFSC   INTCON,INTF
01DF 2097          0662          CALL    Do_I1State      ;Go Do Task #2 - 0 state only
                   0663 I1:
01E0 180C          0664          BTFSC   PIR1,TMR1IF
01E1 2097          0665          CALL    Do_I1State      ;Go Do Task #2 - 1-B states
                   0666
01E2 190B          0667          BTFSC   INTCON,RTIF
01E3 21D1          0668          CALL    Do_Inc_Time     ;Go Inc Time_Bit every 512uS
                   0669
01E4 0824          0670 POP:     MOVF    Temp_FSR,W
01E5 0084          0671          MOVWF   FSR
01E6 0E23          0672          SWAPF   Temp_Stat,W
01E7 0083          0673          MOVWF   STATUS
01E8 0EA2          0674          SWAPF   Temp_W
01E9 0E22          0675          SWAPF   Temp_W,W
01EA 0009          0676          RETFIE                  ;Return from Interrupt
                   0677
                   0678          END
                   0679
                   0680
Errors   :     0
Warnings :     0
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN588

PIC16/17 Oscillator Design Guide

Author: Kim Peck

Designing a clock oscillator without some knowledge of the fundamental principals of acoustic resonators is possible but fraught with the uncertainty of "cut and try" methods. While the oscillator may be made to run with the chosen resonator, it is quite likely that the unit will be slightly off the intended frequency, be grossly off frequency because it is operating on an unintended mode, or have unacceptable temperature characteristics because the wrong resonator was chosen for the application.

This application note is primarily for informational purposes. It is intended to help the designer of clock oscillators understand the parameters of crystal resonators and the terminology of the crystal resonator industry, both of which tend to be somewhat mysterious and arcane to the uninitiated. Details of crystal cuts and rotations, for instance, are of no use to the oscillator designer, only the designer of crystals. The oscillator designer still needs to understand and be able to predict the performance and tradeoffs associated with these parameters. This is not an in-depth or rigorous treatment of acoustic resonators, but a practical guide, which should allow the designer to gain a basic understanding, and to help in choosing and specifying resonators.

INTRODUCTION

The main purpose of the oscillator in the PIC16/17 parts, or almost any other microcontroller, is to provide a reliable clock for the controller processes. At the most basic level, the clock provides a timing interval to account for circuit rise times and to allow data to stabilize before that data is processed. This is a "synchronous" process. The clock also provides an opportunity for the programmer to perform time keeping of several types. In the PIC16/17, the clock also drives hardware dedicated to timekeeping. The applications may include keeping "real time", or timing sensitive processes such as serial data communication. The accuracy of these timing applications is dependent upon the accuracy of the clock oscillator.

Design Challenges

The PIC16/17 controllers offer unique design challenges because they are uniquely flexible. Flexibility usually demands difficult decisions on the part of the designer, but offers otherwise unattainable performance. The multiple oscillator options and wide range of operating voltages require awareness of advantages and tradeoffs of various configurations. The PIC16/17 designer must be able to accurately predict stability performance of various configuration and then obtain that performance from the PIC16/17 clock in order to successfully implement these functions .

Wide Voltage Range

The PIC16/17 operates over such a wide voltage range that the oscillator parameters may be the limiting factor in the operation of the controller. If low power operation at low voltages is desired, the loop gain must be raised in order to insure reliable clock operation. If a nominal supply voltage is available, the loop gain must be reduced in order to prevent excessive power dissipation in the crystal. If battery operation is intended, then a careful balance must be struck between reliable operation at the low voltage, and damaging delicate resonators, or spurious oscillations at the high voltage when the battery is fresh.

Low Power

The outstanding performance of the Low Power option places a burden on the designer who would take advantage of this feature. The frequency chosen must be the lowest practical. Attention must be paid to the reactances associated with the crystal so as not to excessively load the oscillator output and cause excessive power consumption.

Low Cost

The low cost of the PIC16/17 series presents a challenge in finding commensurately low-cost components to complete the design. The relationship between cost and performance when various types of resonators are considered, is far form linear. The low cost of the PIC16/17 microcontrollers, may remove it from the position of being the cost driver in some designs, challenging the designer to aggressively seek cost reductions in components which were previously not considered. The second challenge offered by such economical parts is that of new applications which were not considered practical before the advent of PIC16/17 processors.

© 1994 Microchip Technology Inc.

THEORY OF OSCILLATORS

Conditions necessary for oscillation

An oscillator is a device which operates in a closed loop. This condition can be difficult to analyze, but the techniques for analysis are as valid for motor speed controls as it is for phase lock loops and oscillators. Oscillators are somewhat unique in that they are intentionally unstable, but in a controlled manner. In order for oscillation to occur in any feedback system, two primary requirements must be met. The **total** phase shift must be zero or 360 degrees at the desired frequency and the system gain must be unity or greater at that frequency.

The Ideal Oscillator

The ideal oscillator is a perfectly flat temperature coefficient, is 100% power efficient, has no limits on operating frequency, has no spurious modes, has a perfect output wave shape, and is available in the high degrees of miniaturization which exists in semiconductors. This oscillator of course, does not exist. The primary limiting factor for most oscillator parameters is the resonator. The following is a discussion of the trade-off, potential advantages and primary disadvantages of several popular types of resonators, and how they will behave in a PIC16/17 oscillator design.

RESONATOR BASICS

There are several types of resonators available to the designer of microprocessor clocks. They all provide trade-offs between performance, size, frequency range and cost. Resonators for clock oscillators usually fall into two basic groups. These are quartz and ceramic resonators. Historically, ceramic resonators came into use in oscillators much later than quartz crystals and derive all of their terminology and conventions from the longer history of quartz crystals. A third type of clock oscillator is the RC (resistor / capacitor). This oscillator is a relaxation type, and employs no resonator as such. While this type requires the same basic conditions for oscillation to occur it is better described using different techniques and analogies.

Quartz Resonators

Quartz is the crystalline form of silicon dioxide. This same material, in amorphous form, is commonly found as beach sand and window glass. As a crystal, it exhibits piezoelectric effects as well as desirable mechanical characteristics. A quartz crystal resonator is an acoustical device which operates into the hundreds of MHz. Its resonance and high Q are mechanical in nature, and its piezoelectric effects create an alternating electrical potential which mirrors that of the mechanical vibration. Although it is one of the most common of naturally occurring crystals, natural quartz of sufficient size and purity to be used in the manufacturing suitable resonators, is unusual and expensive. Almost all modern resonators are manufactured using cultured quartz, grown in large autoclaves at high temperatures and pressures.

Whether naturally occurring or cultured, quartz crystals occur as six-sided prisms with pyramids at each end. This raw crystal is called a "boule". In an arbitrary coordinate system the Z , or optical axis, runs the length of the crystal, connecting the points of the pyramids at each end. If one views this hexagonal bar on end, three lines may be drawn between each of the six opposing corners. These are called X axes. Perpendicular to each X axis is a Y axis, which connects opposite pairs of faces. When the boule is cut into thin plates or bars called blanks, the cut of the saw is carefully oriented either along, or rotated relative to one of these axes. Orientation of the saw is chosen based on the mode of vibration for which the plate is intended, and the desired temperature profile. Plates are usually rounded into discs. Types of crystal cuts are named for the axis which the cutting angle is referenced when the blanks are cut from the boule. After being cut and rounded, the blanks are lapped to frequency and any surface finishing or polishing is done at this time. Electrodes are deposited on the blanks by evaporation plating, and the blank is mounted in the lower half of the holder. It is finished to the final frequency by fine adjustments in the mass of the electrode plating, either by evaporation or electroplating. The top cover is then hermetically sealed by one of several methods, which include cold welding and solder sealing.

Most crystals made today are A-T cut, which employ a thickness mode. This mode provides the highest frequency for a given thickness of the plate, and the best possible frequency stability over most temperature ranges. Many other modes of vibration are possible. Flexure modes are usually bar shaped, and are used for low frequency (near 100 KHz) resonators. Tuning fork crystals are a special case of this type.

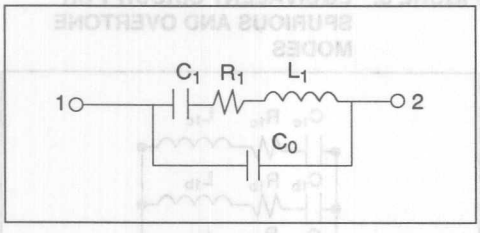
© 1994 Microchip Technology Inc.

Ceramic Resonators

Unlike quartz resonators, which are cut from a single crystal, a ceramic resonator is molded to a desired shape instead of grown. The material is polycrystalline form of barium titanate, or some similar material. The electrical model is almost identical, with the addition of one resistor, as the material is intrinsically conductive. The material is artificially made to exhibit piezoelectrically active by allowing it to cool very slowly, as in growing a quartz crystal (not nearly as long a time), but in the presence of a strong electric field. The molecular electric dipoles align themselves with the applied electric field. When the material has cooled, the alignment of the electric dipoles is retained, which is equivalent to piezoelectricity.

These materials have elastic properties that are not as desirable as quartz, and so their performance is not equal to that of quartz resonators. Specifically, ceramic resonators have far lower Qs and frequency deviations due to temperature on the order of 1000 to 10000 times greater than that of an A-T cut quartz crystal. The cost of ceramic resonators is much lower however, because the material is not grown under the extreme and expensive conditions that are necessary for quartz. They are also much smaller than A-T cut quartz resonators, particularly at frequencies under 2 MHz.

FIGURE 1: RESONATOR EQUIVALENT ELECTRICAL CICUIT



Since the "Q" of ceramic resonators is generally lower than quartz, they will be more easily pulled off frequency by variations in circuit or parasitic reactances. This is desirable if a circuit is designed with a variable element, as greater tuning range is realized. It is not desirable if the highest possible stability is the design goal, because the resonator will be more susceptible to variation in parasitic reactances, such as capacitors formed by circuit board etch, and temperature variations of intended circuit reactances. These variances will add to the already substantial deviation over temperature of the resonator itself. If your stability needs are modest however, ceramic resonators do provide a good cost / performance trade-off.

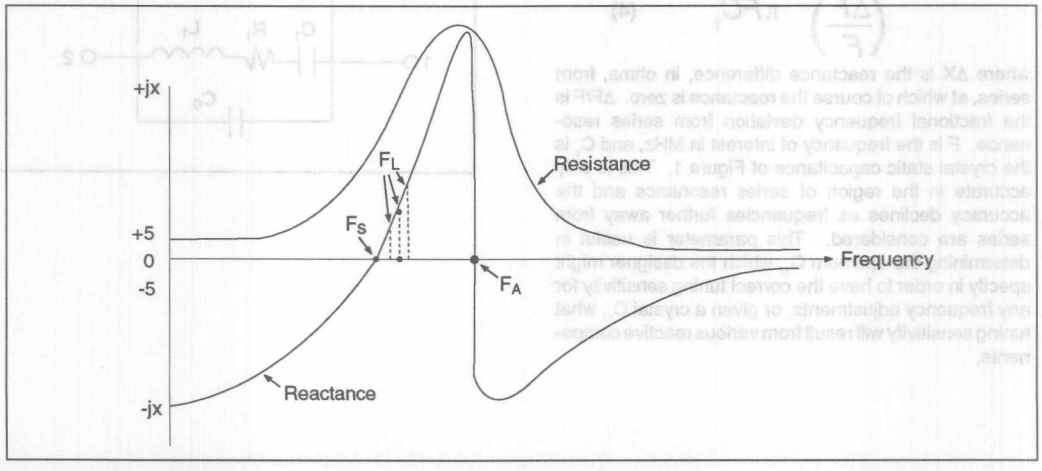
Equivalent electrical circuit

The circuit shown in Figure 1 is a close approximation of a quartz or ceramic resonator. It is valid for frequencies of interest to the PIC16/17 designer. Not all of the parasitic element are shown as they are not important to this discussion. In this circuit, L_1 and C_1 are the reactances which primarily determine the resonator frequency, while a series resistor represents circuit losses. A shunt capacitor, C_1 represents the holder and electrode capacitance.

Because L_1 and C_1 are associated with the mechanical vibration of the crystal, these are commonly referred to as the **motional parameters**, while C_1 is called the static capacitance. The reactance of L_1 and C_1 are equal and opposite at the **series resonant frequency**, and their magnitude is very large as compared to R_1. The phase shift at the series resonant frequency is zero, because the reactances cancel. The series resonant frequency is calculatedas showm in equation 1.

$$F_s = \frac{1}{2\pi\sqrt{L_1 C_1}} \qquad (1)$$

FIGURE 2: REACTANCE VS. FREQUENCY PLOT



© 1994 Microchip Technology Inc.

The actual series resonant frequency as determined by the zero phase point is slightly lower than this calculation because of the effects of C_0, and for practical purposes can be considered identical. This fact may be useful to those designing tunable crystal oscillators. These resonator parameters are generally considered to be constant in the region of the main resonance, with the exception of R_1. A plot of reactance over frequency is shown in Figure 2. The point labeled F_s is the series frequency, while F_L, is the frequency where the crystal is resonant with an external load capacitor. Operation at this point is sometimes called parallel resonance. F_A is the frequency where the crystal is anti-resonant with its own electrode capacitance. Only the region below F_A is useful as an oscillator. Notice that the resistive component begins to rise, before F_s and continues steeply above F_s. This makes operation with small load capacitors (large reactances) difficult. One must be sure that if the resonator is specified to operate at a load capacity that the maximum value of R_1 is specified at that operating point. The zero phase shift point is the most common method of identifying the exact series resonant frequency. When the series frequency is known, operation at a load reactance is easily calculated as follows:

$$\frac{\Delta F}{F_s} = \frac{C_1}{2(C_0 + C_L)} \qquad (2)$$

where ΔF is the deviation from F_s to F_L, F_L is the operating frequency when in series with a load capacitor, F_s is the series resonant frequency (without any load capacitor), and C_L is the load capacitor.

The value of R_1 at the frequency F_L can be approximated by:

$$R_1 = R_L\left(\frac{C_L}{C_0 + C_L}\right)^2 \qquad (3)$$

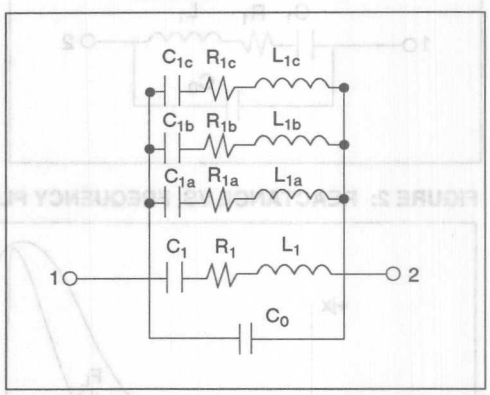
The reactance slope in the region of the series resonance can be approximated by:

$$\left(\frac{\Delta X}{\frac{\Delta F}{F}}\right) \approx \frac{10^6}{\pi F C_1} \qquad (4)$$

where ΔX is the reactance difference, in ohms, from series, at which of course the reactance is zero. $\Delta F/F$ is the fractional frequency deviation from series resonance. F is the frequency of interest in MHz, and C_1 is the crystal static capacitance of Figure 1. This is only accurate in the region of series resonance and the accuracy declines as frequencies further away from series are considered. This parameter is useful in determining the optimum C_1, which the designer might specify in order to have the correct tuning sensitivity for any frequency adjustments, or given a crystal C_1, what tuning sensitivity will result from various reactive components.

The ratio of the reactance of L_1 or C_1 to R_1 is arbitrarily designated as Q. This is also known as the quality factor, and applies to any reactive component. The series resonant frequency of the crystal is the sum of the total series reactances. Quartz A-T cut crystals exhibit spurious modes which are always found at frequencies just above the main response. These are always present and are not associated with activity dips. There are also odd ordered mechanical overtone modes. Any of these modes (spurious or overtone) can be modeled as duplicates of the primary RLC electrical model, and placed in parallel with it (see Figure 3). Notice however, that there is only one C_0. Near the resonance of each series circuit, the effects of the other resonances may be considered negligible. Each resonance of course, has its own motional properties, the one of primary interest here is the R_1 of each resonance. The R_1 usually increases with increasing overtones, making the higher overtones more lossy. The PIC16/17 designer must take care to specify crystal spurious to always be of higher resistance than the desired response. This can be achieved in a well designed resonator. A heavy metal such as gold, as an electrode, will discourage higher overtones, by virtue of its higher mass. Crystals designed for high frequencies, almost always use a lighter material, such as aluminum. Electrode size also plays an important role.

FIGURE 3: EQUIVALENT CIRCUIT FOR SPURIOUS AND OVERTONE MODES



© 1994 Microchip Technology Inc.

OSCILLATORS

Phase and Gain

As stated earlier, two conditions must be met for oscillation to occur. The phase shift must be zero or 360 degrees at the desired frequency, and the total system gain must be one greater or at that frequency. Logic gates or inverters are convenient for this purpose. They have large amounts of gain, they limit cleanly, producing square waves, and their output is appropriate for directly driving their respective logic families. Most oscillators in this family use an inverting amplifier, as shown in Figure 4. The phase shift is 180 degrees through the gate, and the two reactances at either end of the crystal are chosen to provide an additional 90 degrees each, bringing the total to the required 360 degrees. The primary effect of changes in phase is to shift the operating frequency (to tune the crystal). The primary effect of changes in gain is to cause the oscillator to cease functioning when reduced, or cause spurious modes and excess power to be dissipated in the crystal when increased.

Oscillation will occur at the frequency for which the total phase shift is 360 degrees. This is true for any frequency (or resonator response) for which the gain is greater than unity (including unwanted responses). The series resistor (R_s) is used to adjust the loop gain, and to provide some isolation from reactive loads for the amplifier. The lower limit of loop gain is determined primarily by the need for sufficient excess gain to account for all variations, such as those caused by temperature and voltage (not just in the amplifier, the crystal resistance may change as a function of temperature). The upper limit of loop gain should be that where it becomes possible (or at least likely) for the oscillator to operate on a spurious mode. In some resonators damage to the resonator is the overriding concern regarding drive level. If the stability requirement is rather "loose" the stability problems may not be the first indications of trouble. Excessive drive levels in tuning fork types for instance, may cause damage to the point that the crystal unit fails. It is important to estimate drive levels before operation begins, include and adjust a series resistance appropriately, and by measurement, verify the results.

Estimating Drive levels

The drive levels may be estimated with the following steps. First find the load impedance presented by the crystal network, including the phase shift capacitors and the amplifier input impedance. This is found by the following:

$$R_n \simeq \frac{X_C^2}{R_S + R_{OSC1}} \quad (5)$$

where R_N is the network impedance. X_C is the reactance of one phase shift capacitor (assuming they are the same). R_{OSC2} is the input impedance of the OSC1 pin (should include reactance). R_S is the reactance + the resistance at the operating frequency ($R_S + jX_S$).

The current delivered into this impedance is found by:

$$I_n \simeq \frac{V_{OUT}}{R_S + R_N} \quad (6)$$

where I_N is the RMS current drawn by the network. V_{OUT} is the OSC2 output RMS voltage. R_N is calculated above. R_S is described above. The current which passes through the crystal then is found by:

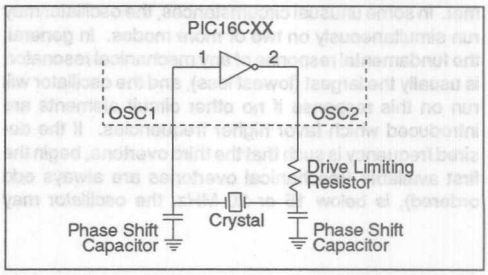
$$I_S \simeq \frac{V_C}{R_S + R_{OSC1}} \quad (7)$$

The power dissipated by the crystal is then found by I_S squared times the crystal R_1.

Controlling Drive Levels

When designing any oscillator, one should take care not to lower the loaded Q of the resonator by inserting any resistive components between the phase shift capacitors (or any other reactive components) and the crystal. If It is necessary to reduce the drive level to the crystal, or lower the overall loop gain, resistance should be inserted between the amplifier output, and the crystal (see Figure 4). This method is much better than changing load reactances, which will have no significant effect on gain until the frequency has been pulled well away from the design center. This will also have the more significant effect of raising the operating current, because if no series resistor is present, the larger reactance of the phase shift capacitor will load the OSC2 output directly. If a very low drive level is required, such as with tuning fork type crystals, the series resistor is the best method. The resistor should be adjusted until the unit just runs with a typical crystal at the lowest operating voltage, and the resulting drive measured at the highest operating voltage. The actual resistor value is best determined experimentally with a representative sample of crystals, and a broad range of values should be satisfactory. In general, the point where oscillation stops for any crystal unit (which is within specified parameters), is the resistors upper limit. The lower limit may be zero ohms, for a less fragile crystal type, depending on the operating frequency. If no spurious or overtone modes are encountered, it is likely that the oscillator may have relatively little excess gain at that operating frequency. If the resulting drive level at the higher voltage is still unacceptable, then the supply voltage variations must be reduced.

FIGURE 4: PIC16/17 OSCILLATOR CIRCUIT



5

© 1994 Microchip Technology Inc.

Measuring Drive Levels

Drive levels **cannot** be easily measured with any certainty by reading voltages at each end of the crystal. This is because of the phase shift which is present in varying degrees, depending on how close to series resonance of the crystal, the oscillator is operating. It is much more reliable and accurate to measure the crystal current with a clip-on type oscilloscope current probe. This probe may require an outboard amplifier in order to measure very low drive levels. It is also important to accurately know the series resistance of the crystal under the same operating conditions of frequency and drive level. This information is easily obtained with network analyzer or a modern crystal impedance meter. While the oscillator designer may not be equipped with such a meter, the manufacturer of the crystal most certainly should be, and the resistance data should be provided for at least one, and perhaps several possible drive levels, if variations in drive are expected.

UNDESIRED MODES

Mechanical resonators are not perfect devices. They exhibit many spurious responses, either continuously or over narrow temperature ranges. If a quartz resonator is swept with a R.F. network analyzer, several smaller responses will be seen just above the main response. These are always present in mechanical plate resonators. For oscillator applications, they must be specified to have a lower response than the desired mode. The crystal designer can control these to some extent by varying the plate geometry and the electrode size. These spurious modes are usually similar in nature to the main response, and do not vary in relation to it to any important degree. Other spurious are caused by completely different modes of vibration, and have radically different temperature curves. These may lay unnoticed until a temperature is reached where the two temperature curves intersect. At this one temperature, the spurious mode traps some of the mechanical energy created by the main mode. This causes a rise in the series resistance, usually accompanied by an unacceptable change in frequency. With a very small change in temperature, the effect will disappear. This is know as an "activity dip", activity being a dimensionless mechanical property which is inversely proportional to resistance. These can also be successfully specified away in most resonators. Any response of the resonator, be it spurious, or mechanical overtones, may control the oscillator output frequency if phase and gain criteria are met. In some unusual circumstances, the oscillator may run simultaneously on two or more modes. In general, the fundamental response of any mechanical resonator, is usually the largest (lowest loss), and the oscillator will run on this response if no other circuit elements are introduced which favor higher frequencies. If the desired frequency is such that the third overtone, begin the first available (mechanical overtones are always odd ordered), is below 15 or 20 MHz, the oscillator may occasionally run at around three times the desired frequency. This may only happen every third or fifth time the unit is activated. The unit may start correctly, but jump to the higher overtone when the unit is exposed to a very narrow temperature range, but remain there after the temperature has changed. The best fix for this problem is usually a reduction in overall loop gain. Occasionally a crystal may have a very low resistance at overtone modes as well as the fundamental. In this case it may be useful to specify overtone modes, as spurious and guarantee at least a -3db difference between the overtone and the fundamental responses. This condition will already exist for 99% of the resonator designs, and is not usually specified.

It is also best not to insert any large reactances which would compete with the Q of the crystal for control of the oscillator output frequency. If this is done (say, for the purpose of adjusting the frequency of the oscillator), the tuning reactance, (usually a variable capacitor), must be accompanied by an equal reactance of the opposite sign in order to bring the total loop reactance back to zero (unless the crystal is designed to operate with that large series reactance, which could cause other problems). If the oscillator is pulled far enough from the series frequency, the rising crystal resistance will lower the loaded Q of the crystal until the reactance slope of these components will compete with that of the crystal. This will cause the oscillator to "run " on these components instead of the crystal, the loop being completed by the C_0 of the crystal. The component with the steepest reactance slope will control the frequency of the oscillator. The tuning sensitivity of these components will also be directly proportional to the magnitude of their reactances. Any unwanted variation of these components will have increased consequences for the stability of the oscillator. Another source of spurious is a relaxation mode which is caused by the amplifier bias circuits and the phase shift capacitors. The loop is completed through the crystal C_0. Again, a series resistor will usually solve this problem, although in some cases the amplifier bias values may need to be changed.

Load Capacitors

In gate or logic type of oscillator, the crystal is usually manufactured to be slightly inductive at the desired frequency, and this inductance is canceled by the two phase shift capacitors. **The primary purpose of these capacitors is to provide the phase shift necessary for the oscillator to run.** Their actual value is relatively unimportant except, as a load to the crystal, and as they load the output when no series resistor is used. These reactances are the sum total of selected fixed capacitors, any trimmer capacitors which may be desired, and circuit strays. If a loop is considered from one crystal terminal through one phase shift capacitor through ground and the second phase shift capacitor, to the second crystal terminal, all the reactances including the crystal motional parameters must add up to zero, at the desired operating frequency.

© 1994 Microchip Technology Inc.

As a crystal load , all circuit reactances external to the crystal should be thought of as a series equivalent. In order to know the total load reactance seen by the crystal, the total shunt reactances on either terminal are summed, and the series equivalent is calculated. This should include the OSC1 and OSC2 terminal reactances, but these are negligible if they are sufficiently small when compared to the phase shift capacitors. The value of these capacitors, is then chosen to be twice the specified load capacity of the crystal. It some adjustment of the frequency is necessary, one of the phase shift capacitors can be chosen at a smaller value, and the difference made up by a variable capacitor placed across it. An alterative method is to place a larger value of trimmer capacitor in series with the crystal. The value of the trimmer capacitor must be chosen along with the phase shift capacitors, all in series, to give the correct load capacity. Frequency should not be adjusted by shunting the crystal with a capacitor. If it is desired to use a crystal which is finished at series resonance, an inductor of equivalent reactance to half of the phase shift capacitors, must be placed in series with the crystal.

STABILITY

General

Frequency stability is the tendency of the oscillator to remain at the desired operating frequency. Its deviation from that frequency is most conveniently expressed as a dimensionless fraction, either in parts per million (PPM) or a percentage. Absolute deviations in Hz, must always be referenced to the operating frequency, which is less convenient and not universal. In the following discussion of temperature characteristics, one can see that the fractional deviations are universal without any direct effect of operating frequency. In order to calculate a total frequency stability, various separate elements must be identified and quantified. Not all parameters of frequency stability are important to each design. The various items which effect the frequency of an oscillator are: the temperature profile of the resonator, the resonator's room temperature frequency tolerance (also known as "make tolerance"), its long term frequency drift which is normally know as ageing, and its sensitivity to other circuit reactances. Is it possible to adjust it to the exact desired frequency? If not, how big is the error due to other component tolerances. Due to the complexity of this combination, most crystal manufacturers will offer a standard crystal which is guaranteed to be ±100 PPM over -20°C to + 70° C, or ±30 PPM over -0°C to + 60°C. Note that the temperature coefficients of some of the curves in Figure 2 are much smaller than this over the same temperature range. Large portions of these tolerances are devoted to make tolerances and circuit component tolerances. The room temperature items can be relatively simple to specify in the resonator design. If careful attention is paid to specifying the crystal, or designing the oscillator to accommodate a standard crystal, more of the total stability requirement can be devoted to the temperature profile, or the overall stability requirement can be reduced. The temperature profile, however, is subject to other circuit influences external to the resonator. These may be somewhat more difficult to perceive and control. If, for example, the chosen resonator is an A-T cut or tuning fork type, possessed of a nominal temperature profile of less than 50 PPM over the desired temperature range, external influences, such as capacitor temperature coefficients, may play an important part in the overall stability of the oscillator. If however, a ceramic resonator is chosen, it's temperature profile of 40 to 80 PPM/C, will dominate the oscillator stability, and 5 or 10 PPM shift from changes in amplifier impedance or capacitor temperature coefficients will not be important. The designer may chose a crystal even when the overall stability specification (of the oscillator) does not require it, giving large design margins. If any amount of testing or adjustment of the oscillator frequency is needed with the lower cost resonator, the crystal may be more cost effective. When designing any resonator as part of a simple logic type oscillator circuit (Figure 4), some attention should be given to swapping the amplifier reactances (that is to make them a very small part of the sum total circuit reactance) with the phase shift capacitors, and any other circuit reactances. This is at least, a good design practice. The largest reactance has the most effect on the operating frequency. It follows then that the motional parameters, which have very large reactances, dominate the equation for the total reactance, and so the operating frequency of the oscillator. Another good design practice, is to specify only as much pullability as is required to accommodate the make tolerance and ageing of the resonator, and tolerance of other circuit elements. Pullability is a function of the ratio of C_1 to C_0. As the reactance of the crystal C_1 increases it becomes more stable in relation to outside reactive influences. It also becomes more difficult to intentionally adjust its operating frequency. If too high a C_1 is specified, the resonator will be sensitive to external influences, and the effect of these influences may be as large or larger than the temperature profile. If the C_1 is to small, it may not be possible to adjust the unit exactly to the desired operating frequency. The small electrode size needed to realize a low C_1 may also concentrate the mechanical energy in a very small percentage of the blank, causing unpredictable behavior. In order to quantify pullability in terms of C_1 to C_0 ratio and load capacitance (refer to the Equivalent Electrical Circuit section).

5

© 1994 Microchip Technology Inc.

A-T Cuts

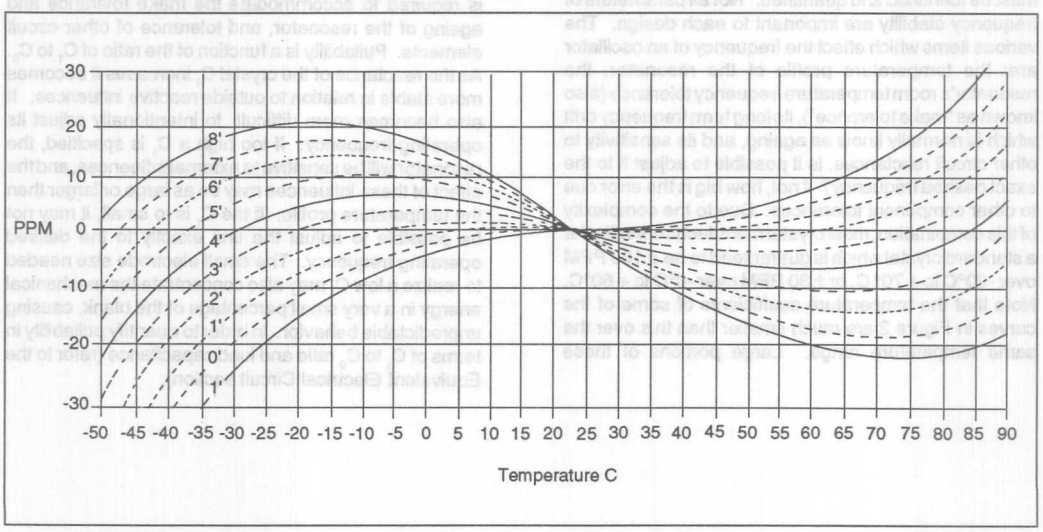
The A-T cut crystal and its variations, is by far the most popular resonator in the world today. A-T cut crystals are popular because the "S" shaped temperature curve is centered very near room temperature, typically around 27 C. This temperature profile is compact, symmetrical, and most manufacturers are able to provide good control of the cut angle.

Because most of the crystals manufactured in the last 40 years have been A-T cuts, they are very well understood and documented. This is important because while temperature coefficients can be calculated from the mechanical properties, such as elastic constants, they can (and have been) measured with much more accuracy. When the temperature coefficients are accurately known, the temperature profile can be calculated for an individual set of conditions. Figure 5 is a family of temperature curves of A-T cut crystals used for this purpose. Each curve represents a possible crystal at incremental changes in the cut angle. The practical limit for accuracy of the cut is about ± 1 minute of angle, and in any lot of crystals there will be variations of about ± 1 minute. The designer will create a box around these curves using the desired temperature limits as the vertical sides, and the desired frequency tolerance for the horizontal lines, as shown in Figure 5. If the curves are spaced at intervals of one minute of angle, then the specification is a practical one if three of these curves (± 1 minute) fit within the outlined area. It is possible to purchase crystals with a closer tolerance, but this is mostly a matter of yields, rather than a better process. The steeply increasing cost will reflect the higher reject rate.

When purchasing a crystal, do not attempt to specify a specific angle, rather specify a frequency deviation between turning points, with tolerances. The mathematics of these curves, is represented by a linear term between two turnover points, whose inflection point is at or near 27° C. The temperature above the high turnover and below the lower turnover, are characterized by cubed terms (very steep). This was described by Bechman in the late 1950s as a third order polynomial. This can be seen in Appendix A. Notice that as the linear portion of the curves between the turnover points approaches zero slope, the turnover points move closer together. This tends to limit the temperature range over which very small stabilities can be realized. If the required operating temperature range is inside of the range of the turnover points, a low angle is desirable. If so specified, most manufacturers will provide a crystal with temperature profiles on the order of ±5 to ±10 PPM over modest temperature ranges for a reasonable cost. If the desired operating temperature range is outside of the range of the turnover points, a higher angle is desirable in order to keep the frequency at the extreme temperatures within the same realm as the deviation between the turnover points. This may approach ± 60 PPM for large temperature ranges, but is still far less than the smallest deviations achievable with other resonators over the same temperature range.

What is not immediately obvious, is that if a linear frequency shift with temperature is applied to a frequency curve, the result is a rotation of the curve which will eventually match another member of the curve family. There is no other distortion of the temperature curve if the frequency shift is linear, such as from a

FIGURE 5: FREQUENCY VS. TEMPERATURE CURVE FOR A-T CUT CRYSTAL



© 1994 Microchip Technology Inc.

temperature compensating capacitor. This fact also gives a convenient graphical technique to estimate the effect of temperature coefficients of other components. There exist several flaws in this picture of the A-T cut temperature profile, which may prevent the PIC16/17 designer from completely realizing the stability suggested by the curves in Figure 5.

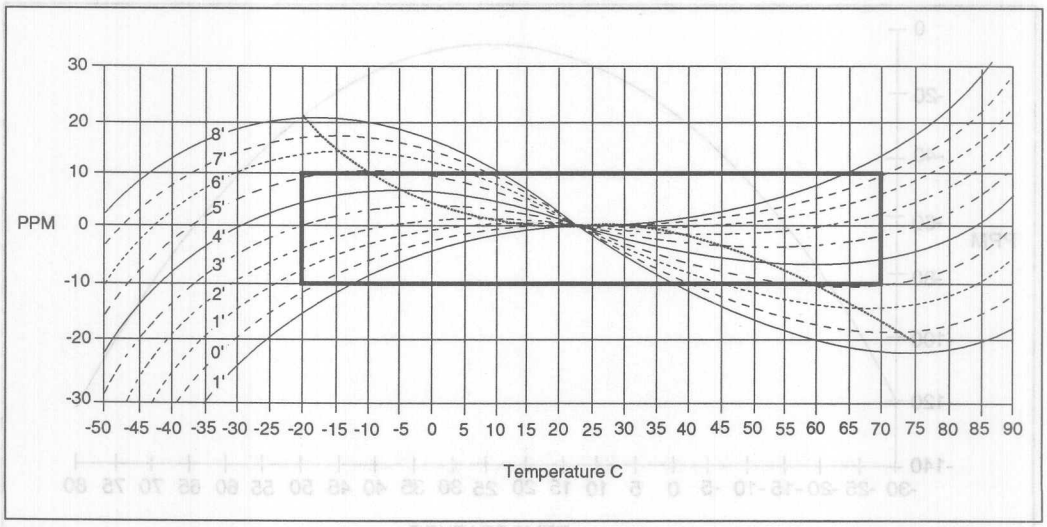
The first problem which may arise when choosing a crystal angle based upon these curves, is that there may occur some rotation of the crystal angle due to external circuit influences. The most common influences are that of reactive components (inductors and capacitors). Most inductors have a slight positive temperature coefficient, while capacitors are available in both positive and negative temperture compensating types. Non-compensating type capacitors vary greatly depending on the dielectric from which they are manufactured. The best capacitors for frequency determining elements, are ceramic types with NP0 (flat) temperature coefficients. Avoid at all cost, capacitors made from Z5U material. These have a large temperature coefficient and are unsuitable even for supply line decoupling or D.C. blocking capacitors. This is because a slight change in the R.F. impedance which shunts the VCC and VDD pins, will have an effect on the output impedance of the amplifier, and so an effect on frequency. The effect will be on the order of a few PPM, and may well be of secondary importance, depending on the stability requirement. A word about D.C. voltages and crystals. It is permissible to place a D.C. voltage across the terminals of the crystal. This does cause a small change in frequency, but that change is not significant for stabilities of ±5 PPM or greater.

The second problem is one of dynamic temperature performance. When the unit has stabilized at any temperature on the curve, the frequency will agree with the curve. While the temperature is slowing however, the frequency may be in error as much as 5 to 15 PPM depending on the temperature change. This effect is caused by mechanical stresses placed on the blank by temperature gradients. These can be minimized by thermally integrating the crystal, and joining it to a larger thermal mass. One oscillator engineer has been known to attach a block of alumina (ceramic) to both of the crystal pins in order to join them thermally. Any other mechanical stresses placed upon the pins or leads of an A-T cut crystal unit will also result in a dramatic frequency shift (if the unit is not damaged first). This is to be avoided.

The third item which will cause a deviation from the curves of Figure 5, is spurious response. This is know in the crystal industry as an activity dip. This name originates from a time when the series resistance was referred to as crystal activity, and the frequency change is accompanied by a marked rise in series resistance. This phenomenon occurs when mechanical energy is coupled from the normal thickness shear mode into another undesired mode of vibration. Several other modes are possible for finite plate resonators, and they are usually resonant at frequencies well away from the design frequency. These modes will often have radically different temperature profiles, and may intersect with the profile of the desired mode at only one very narrow range of temperatures (much less than 1°C). This makes an activity dip difficult to spot in normal

5

FIGURE 6: FREQUENCY VS. TEMPERATURE SPECIFICATION FOR A-T CUT CRYSTALS



© 1994 Microchip Technology Inc.

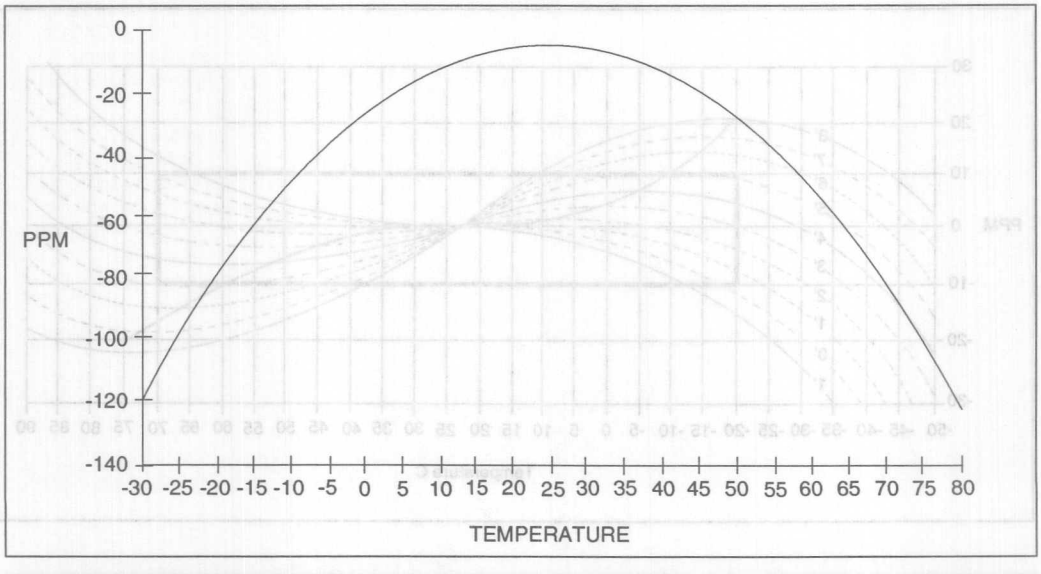
testing. Those which are discovered are often around room temperature where temperature changes are more gradual. This coupling between modes is greatly effected by drive level, and the best crystal may exhibit a dip if grossly overdriven. Fortunately, most manufacturers today can produce a crystal which is free of significant dips if so specified. As the accompanying rise in resistance is occasionally large enough to cause oscillation to halt, the PIC16/17 designer should always specify activity dips to be less than 1 PPM, even if the overall stability requirement is much larger than this. In the interest of low cost and flexibility, the designer may also specify activity dip in terms of a maximum change in resistance.

The other important effect on frequency stability of A-T cut crystals, is ageing. This is the long term frequency shift caused by several mechanisms, the most notable being mass loading of the resonator, causing the frequency to shift ever downward. Because this is the primary mechanism, the cleanliness of the interior of the unit is of prime importance. This is in turn greatly effected by the method used to seal the unit, and the type of holder chosen. If the unit is subjected to excessive drive levels, the frequency may age upwards, indicating electrode material is being etched off of the blank. A good general purpose high frequency crystal using a solder seal holder may be expected to age about 10 to 20 PPM / year maximum. Resistance weld holders will average 5 to 10 PPM / year, and for high stability applications, cold weld crystals are available at ageing rates of 1 to 2 PPM / year. The ageing rates of most crystals will decay exponentially, the most change being in the first year. Ageing rates are different if the unit is operated continuously, but aging will continue even if the unit is not operated.

32 KHz Watch Crystals

The typical 32 KHz watch crystal is a tuning fork type. This is a special case of a flexure mode (N -T cut). The unusual nature of this flexure type is that it is indeed shaped like a tuning fork. This shape gives the crystal a very small size for its low frequency of operation and is almost always manufactured in the NC 38 holder. This is a tube 3mm by 8mm. This type is available at frequencies from 10 to 200 Khz, although 32.768 KHz is by far the most popular frequency. The frequency is of course 2^{15}, which is ideal for time keeping applications, and being so low is ideal for low-power applications. This type is generally less stable than higher frequency A-T types, but much better than ceramic resonators, the primary attraction being the possibility of very low operating power drains. The PIC16/17 LP option was designed with this crystal in mind. It has a parabolic temperature profile of about .04 PPM / (°C) 2. The turnover point of the temperature profile is near 25°C. In order to calculate the change in frequency it is only necessary to square the difference in temperature from 25°C and multiply by .04. The temperature profile is shown is in Figure 7. The C_1 is on the order of .002 pf, which will make design for frequency adjustment possible but not trivial. The make tolerance is usually about 20 PPM at best, making some adjustment necessary for most applications. The series resistance of this type is very high, on the order of 30,000 ohms. It is imperative that care be taken to limit the drive to the crystal. Only a fraction of a ma. of crystal current will damage this unit, possibly causing it to cease oscillation. This is best done with a series resistor between the OSC2 pin and the junction of the crystal lead and phase shift capacitor (see Figure 7). If the frequency is moving upward in a continuous manner, the drive level is probably to high. A portion of this change will be quite permanent.

FIGURE 7: FREQUENCY VS. TEMPERATURE FOR NC38 TIMING FORK TYPE CRYSTAL



© 1994 Microchip Technology Inc.

Ceramic Resonators

Ceramic resonators are the least stable type available other than Resistor / capacitor networks. The temperature profile is a much distorted parabolic function, somewhat resembling that of some capacitors. The temperature coefficient is on the order of 40 to 80 PPM /°C. The typical specified stability for -20°C to + 80°C is ±0.3% (3000 PPM). The C_1 can be as high as 40 pF, making the oscillator extremely vulnerable to circuit influences external to the resonator. The Rs however is on a par with A-T type crystals, at around 40 ohms. The positive features of this type are the small size, low cost, and relative simplicity of designing it into a PIC16/17 part. Because these have a very low Q, the start-up time can be very good, although with the large phase shift capacitors necessary at low frequencies where this would be an advantage, the bias stabilization time will probably dominate the start-up characteristics. If the stability requirements are very modest, this will be a good choice.

R/C oscillators

The PIC16/17 parts can be configured to operate with only a resistor and a capacitor as frequency determining elements. This is a very low cost method of clocking the PIC16/17. The stability achieved this way is at best only adequate if the only thing required of the oscillator is to keep the PIC16/17 marching along to the next instruction. The main effects on stability are that of the switching threshold of the OSC1 input, and the temperature coefficient of the resistor and capacitor.

HOW TO CHOOSE A RESONATOR

Type Tradeoffs

The primary tradeoffs for a designer when choosing a resonator are frequency, size, stability and cost. The lowest cost oscillator is the RC type. This also has the worst stability. The components however tend to be reliable and small, where as resonators are in general larger and have limitations on the amount of physical punishment they can absorb.

A-T cut crystals have the best overall stability and are available in frequencies from 1 MHz to the upper limit of the PIC16C5X part, and in a roughly 0.5 "square package". T-05s and 0.3 "square packages" are available at higher costs, down to a frequency of around 5 MHz A-T cut crystals also have a smaller overall temperature profile which the designer has the best chance of specifying and controlling. Temperature stabilities on the order of ± 10 PPM are possible over modest temperature ranges. The A-T cut can be sufficiently reluctant to move off frequency in response to parasitic reactance changes that it can fully realize these small deviations over temperature. Such is not always the case with other resonator types or incorrectly specified crystals.

Ceramic resonators offer smaller size and slightly lower cost, although in large quantities, microprocessor grade crystals (± 100 PPM) can be competitive. Ceramic resonators will, however, suffer from temperature stabilities in the 0.3% to 0.5% region. This is a significant step down from quartz crystals of any kind.

A designer must choose a resonator which is available in the desired frequency range, has acceptable temperature characteristics, has the lowest cost package which is appropriate for that resonator and is suitable for the mechanical packaging of the oscillator chosen. A-T STRIP resonators are normal A-T cut resonators in which the resonator blank is cut in a long strip rather than a disc, and the electrodes cover a much higher percentage of the quartz blank. A standard A-T cut crystal is a thickness mode resonator, and is usually cut in the form of a disk. The electrodes usually cover only a small portion of the blank. The remainder of the blank not covered by the electrodes, can be thought of as support structure. By removing this support structure, the size of an A-T cut resonator can be greatly reduced. This type of construction violates several rules having to do with thickness to diameter ratios and greatly reduces the overall mass of the blank. This results in reduced performance in the form of slightly less predictable temperature stability, and dramatically reduced power handling capabilities. The A-T STRIPs are generally available up to 20 MHz, depending on the manufacturer.

Tuning Fork type resonators are a type of flexure mode resonators. They are made from quartz and a very small and a cost which is competitive with microprocessor grade A-T cut crystals. Tuning forks have a predictable parabolic temperature coefficient, but any drive power in excess of their very low specified level will deteriorate this quickly.

If stability requirements are beyond what is achievable with a good, A-T cut crystal, the next option is to drive the OSC1 pin with an external oscillator. A good Temperature Compensated crystal(X) Oscillator (TCXO) is expensive when compared to crystal resonators. Stabilities of ± 1.0 PPM over large temperature ranges are common.

Price discounts for volume quantities do not always occur, because each unit must be individually compensated. This varies greatly with the stability and temperature range, and so of course does the price, which in any case will be much higher than any resonator, which the PIC16/17 designer might consider.

5

© 1994 Microchip Technology Inc.

Size and Performance

It is generally true that as a designer considers resonators of smaller size, he or she is faced with decreasing overall performance. Even the A-T cut crystal, which has the best stability discussed here, will become less stable as size is reduced, especially when plate area is reduced in relation to the spot as in a STRIP A-T. One important factor is the themal inertia represented by the mass of the blank. The bigger the blank, the slower it is to follow the changes in temperature. When the blank changes temperature to quickly, it will deviate from the temperature profiles. The frequency will return to this point once the blank has stabilized at the new temperature, but may be well off the profile during a temperature slew. This problem becomes greater as the size of the blank is decreased. Some stability issues are due to the unusual motional parameters associated with certain miniature resonators. Some low frequency types have a C_1 which is larger than the holder capacitance (C_0), making it extremely easy to tune and vulnerable to external influences. Most miniature types have parabolic temperature coefficients which are large enough to make them inferior to A-T cut crystals and A-T STRIP cuts, not only suffer from temperature transient problems as mentioned above, but are it is also difficult to control their cut angle and finish frequency. Tuning forks have a somewhat more predictable if larger temperature profile. Almost all miniature types will not perform well (or sometimes at all) with excessive drive levels. The drive power to the resonator must be controlled, and usually one resistor is sufficient.

Cost and Performance

The lowest cost timing system, of course, is the RC type. This rugged, low cost and small timing system is useful only for the most forgiving timing applications. If all you need is something to keep the PIC16/17 moving, this is a good choice. The next step in cost is most likely the ceramic resonator. Its size is smaller than most A-T cut crystals, but its frequency stability is measured in percent, rather than PPM. A high C_1 also makes it vulnerable to external influences.

Tuning fork types, like all others, vary in price. They may cost less than the ceramic resonator, if a standard frequency is acceptable, or may cost more than an A-T cut if a nonstandard frequency is ordered. Tuning forks usually have a very low C_1 contributing to overall stability and this may actually make it difficult to trim to frequency. Tuning forks have a relatively controllable parabolic temperature curve. A-T cuts have the best overall stability but their cost varies greatly. A-T cuts have an additional advantage in that their temperature profile is the most easily controlled. This offers some flexibility in specifying the angle of cut. A low angle may be order for minimum deviation near room temperature, or a high angle may be ordered to give minimum deviation at extreme temperatures. Standard frequencies and loose specifications for motional parameters will yield cost and delivery competitive with ceramic resonators. Any nonstandard parameters will raise the cost quickly and almost certainly rule out any "off the shelf" part. It will be necessary to specify a nonstandard crystal, if the greatest possible stability is to be "wrung out" of an A-T cut crystal.

Packages

Quartz crystals have a large and mostly obsolete stable of resonator holders to choose from. This is because of the much longer history of quartz crystals. Most of the nomenclature used to describe them, and the technology used to develop them, comes from the MIL-STD system. These include the HC/6, which is about .750" square, and is only necessary to accommodate the lowest frequency A-T cut. The H/C 43 is only about .500" square, and probably accounts for most of the crystal production in the world today. The H/C 45 is still smaller at about .350" square. There are many other standard part numbers which are variations of these, with pins or wire leads, thin version and short versions, and several different methods of sealing the package. Most manufacturers offer their own nonstandard variations of these, as well as clever ways to surface mount them. Most of these variations however, have their origins in the standard H/C parts. The method used to seal the package will have the greatest impact on price and ageing. Solder sealed crystals are usually the least expensive owing to the modest equipment requirements, and simplicity. Resistance weld is slightly more costly, and cold weld is a distant third. This may be changing as more large volume production is implemented with resistance welded packages. Because more exotic (expensive) materials are involved in the cold weld and resistance weld packages before any crystal is mounted in it, is doubtful that this order of cost will change very much. Both solder sealed and resistance welds leave some residue, which over long periods of time contaminate the blank. This causes long term frequency shifts, known as ageing. Cold welded crystals cost more because of more expensive materials which must be used, and expensive tooling (dies) which eventually ware out. Cold weld packages, if assembled in a clean environment, have the potential for the lowest ageing rates. Glass crystal holders have in the past held a slight advantage in ageing over cold weld types, but in the last several years, cold weld techniques have matured to were they have surpassed the glass holder in performance. Some manufacturers, because of the processes in place, may offer glass at a competitive cost. There is nothing wrong with glass holders, but no particular advantage over a modern cold weld package. In any case, the differences in ageing rates will not be important to all but a few PIC16/17 designers. Most ceramic resonators are only available in two or three packages, depending on the manufacturer. The most popular is the dip molded, ranging from 0.3" to 04" square, with some higher frequencies available in lower profiles. Their major size advantage over crystals, if any, will be in height rather than footprint size.

© 1994 Microchip Technology Inc.

Design Examples

A communications device that is designed around the PIC16C5X part and requires that the connecting units have a close timing relationship. Size is not a primary factor, but cost and stability are. A high clock frequency is desired in order to obtain a good sampling rate of the input signal. The PIC16C5X-HS part is selected for a clock frequency of 8 MHz. An X-T Cut crystal is chosen and specified for a maximum frequency deviation of ± 40 ppm over -20° to +70° C. The frequency is too high for a tuning fork type, and the stability is out of the question for a ceramic resonator. An examination of the A-T cut frequency deviation / temperature curves show that a ± 1 minute angle tolerance will give ± 30 PPM frequency deviation over temperature. This leaves 10 PPM for ageing over the five-year life of the product. An A-T STRIP is a choice but at this quantity, the A-T cut in an H/C 43 cold weld type holder, comes in at a lower bid. Since there is space in the assembly, it is chosen.

HOW TO SPECIFY A CRYSTAL

When the PIC16/17 designer chooses a resonator, whether it is a standard or a custom part, a specification while not essential, is an extremely good idea. A clear specification covering all items of form fit and function, will eliminate any possibility of confusion on the part of the manufacturer, and insure that the part will be suitable for the application. The specification should communicate the requirements to the manufacturer and be an instrument by which questionable parts may be measured. The time for discussion with the manufacturer of the resonator is when the specification is being written, not after. The designer must have or gain a knowledge of what parameters will raise the level of difficulty of manufacturing the resonator, and so the cost. Items which effect cost and levels at which these items become an issue, may vary between manufacturers. A typical crystal design sheet is shown in Figure 8. The A-T cut crystal is likely to have the most detailed specification. Other types of resonators will follow this general form with differences being mostly that of omitting many items. This data sheet is likely to become a document in a drawing package for the design of a larger assembly, so the sheet begins and ends with blocks for a drawing number, signoffs, and revisions. The title informs the manufacturer that the crystal is intended for use in an oscillator, as opposed to filters, or other applications.

Motional Parameters

The first item in the crystal design is the frequency and the operating load. This might include series resonance (no load), but the PIC16/17 designer will almost certainly use a value of about 1/2 of the phase shift capacitors, plus any trimmer capacitors which may be added. It is customary to use a standard value here such as 20 or 32 pF, but a nonstandard value is not very difficult given modern manufacturing equipment. The frequencies possible with the PIC16/17 oscillator should not strain the capabilities most manufacturers.

The second item is the "Make Tolerance". This the accuracy to which the crystal is manufactured at room temperature. This should be at least as small as the temperature deviation, and as small as ± 20 PPM should not effect the cost significantly. Avoid tightly specifying this value. Tolerances of ± 10 PPM and less are quite practical but more difficult and will impact cost. If the stability budget does not allow this for at least ± 20 PPM for tolerance of the crystal and associated components, then an adjustable component may be necessary. The added cost of the parts and labor to adjust them must be weighed against the cost of tighter make tolerances on the crystal. This decision must be made on an individual basis.

The third item in the design parameters is the mode of vibration. This will be the fundamental mode for almost all PIC16/17 designers. Other possibilities include the third overtone operation, but many other parts must be added in order to insure operation on only the desired overtone. While there are some advantages to overtone operation, almost all PIC16/17 designers will specify the fundamental mode. Still, what may be obvious to the PIC16/17 designer must be conveyed to the crystal designer, and so this item should not be omitted, or minimized. Series resistance is usually a "not to exceed" value. A good fundamental mode crystal in the PIC16/17 operating frequency range will not be above 10 or 15 ohms, although the oscillator may run with a higher value. This depends on the frequency and excess gain available from the particular model of the PIC16/17 part, at that frequency. The higher resistance will mean more power dissipated in the crystal, and for this reason a nominally lower value should be adhered to. The load capacity will have an effect on this value. The practically achievable series resistance will rise as the load moves the operating point away from series resonance and towards antiresonance (see Figure 7).

The motional capacitance, or C_1, may be the most troublesome item for the PIC16/17 designer to specify. This item will have the single largest effect on the tuning sensitivity (intended or unintended) of the oscillator. Additionally, if the C_1 is specified to be to small, the crystal designer, who controls C_1 by adjusting the electrode size, will use a very small electrode. This will result in the drive power being dissipated by a small portion of the crystal blank, making drive related areas, such as activity dips and other spurious, more critical. If a large C_1 is specified, the unit may be unnecessarily less stable. The static capacitance or C_0, is usually a "not to exceed" value, and it is not of much interest the PIC16/17 designer unless a large and specific degree of adjustability is required from the oscillator. This may be important if an electrically tunable oscillator is desired. In this case, a specific ratio of C_1 to C_0 could be specified. This would not be a low-cost item. It is customary to leave this blank if a specific value is not desired, or the words "as required" to be placed in that location.

© 1994 Microchip Technology Inc.

Other alternative methods of specifying C_1 and C_0 might include a fractional frequency deviation between series and load capacity operation. The drive level should indicate the highest drive level which the PIC16/17 designer estimates the crystal to experience in operation. The crystal designer would like this value to be very low, rendering it a nonissue. The PIC16/17 designer must specify a practical maximum value and take steps to insure that it is not exceeded. Operation at spurious modes and activity dips are just two of the possible consequences of excessive drive levels. Activity dips are not to be tolerated if reliable operation is expected. To quantify this, a maximum allowable value is placed on the frequency deviation. This should be less than 1 PPM and must be less than 5 PPM (at reasonable drive levels, i.e. less than 1 mW). The maximum permissible drive level is determined somewhat by the size of the blank, the electrode, and therefore the size of the holder chosen. The crystal manufacturer should offer a realistic value.

Temperature Characteristics

The crystal manufacturer must know the temperature range over which operation is intended, and this is the first item under the heading of temperature characteristics. The temperature profile of an A-T cut crystal is controlled by the angle of cut. The desired profile is also chosen in terms of angle. When purchasing a crystal however, do not attempt to specify an angle. These angles are referenced to the crystal's atomic lattice, and are calibrated using an X-ray diffraction technique. Their is little direct correlation between manufacturers. Once again the measured results, in the form of measured temperature profiles, are much more accurate, and are the final word in any process control. The PIC16/17 designer should specify a fractional frequency deviation with tolerances between the turning points of the temperature profile. This is the accepted industry standard for specifying temperature performance, and any crystal manufacturer will readily accept it if the tolerances are realistic. A typical temperature profile might read ; "turn to turn 5.5 PPM + 5.0, - 3.5".

Packages

The type of package and the method of sealing it should be specified, although it may be useful on occasion to leave this item blank. Some manufacturers are equipped especially well for a particular type of holder or sealing process, and may offer a better package than required, at a competitive price. In general however, it is best to specify this at the outset. Package types (holders) greatly effect the price of a crystal. The primary performance effect is that of ageing, though other factors, such as thermal characteristics may also be effected.

Other Resonators

Specification of the NC-38 type crystal is limited to motional parameters, as it is not a "rotated" cut. There is little the crystal designer can do to alter the temperature profile. Specifying a ceramic resonator is mostly a matter of custom frequencies, but some control of motional parameters and package variations are possible, though not common. The large temperature profile tends to dominate all other considerations.

Crystal Example

The following is a specification for a 10 MHz A-T cut crystal which the PIC16/17 designer is likely to choose for a high stability application. The frequency tolerance is ± 20 PPM. A rather modest C_1 of .028 pF ± 20 % is specified and the C_0, though not specified, will be around 5 to 7 pF. It is required to operate on frequency with a 32 pF load. The maximum drive level is 1 mW, and no activity dips greater than 3 PPM will be accepted. The crystal will operate over the temperature range of -20°C to + 70°C. The frequency deviation between turnover points, is 5.3 + 4.5, - 3.2 PPM. Notice that the turning points do not necessarily fall within the operating temperature range (see Figure 5). These deviations between turning points correspond to 1, 2, and 3 minutes of angle relative to the zero coefficient angle. Although the 1 minute curve displays a smaller deviation between the turn points, if it were the center of the angle range the lower end of 0 minutes would be unacceptable due to the rapid changes at the ends of the operating range, where cubed terms are in effect. The exact deviations were computer generated for of each crystal angles. These offer more detail and accuracy than is possible with graphical techniques. An alternate method of specifying this is to set a total deviation over the entire operating temperature range of about ± 8 PPM. This is not as exact, and leaves the manufacturer more freedom to interpret your requirements. If the PIC16/17 designer is not comfortable with these concepts, this may be the best approach. One may notice a small dissymmetry in the turning deviations. This is because the chosen operating temperature range is symmetrical about 25°C, and the inflection temperature of the A-T cut is closer to 27°C or 28° C (see Figure 5). In some cases the center of the operating temperature range may be very different form the inflection point of the crystal, and in order to realize the benefit of the best angle, a frequency offset at 25°C would be needed to center the temperature profile (this would not be part of the design sheet). The package is chosen to be an H/C-49 type which has a resistance weld seal. A maximum ageing rate of 2 PPM / year is required. No unusual shock or vibration is expected for this unit. Under the area of testing, temperature testing is required only on a sample of the lot. All units will be exposed to a thermal shock , and 10 days of ageing at 85 C. Ageing is of concern, so gross leak is specified to be tested on all the units, and a fine leak test is to be performed on a 13% sample. Any notes about the application or special concerns would complete the crystal design sheet.

© 1994 Microchip Technology Inc.

THE PIC16/17 ON BOARD OSCILLATOR(S)

The PIC16/17 device actually contains four complete oscillators which can be selected during the programming process. The selected oscillator is connected to the OSC1 and OSC2 pins, as well as the chip clock drivers by CMOS switches. In the windowed parts, these are all available to the programmer, while the OTP and QTP parts are pre-configured at the factory, and must be ordered as the desired type. The four types of oscillator available in the PIC16C5X / 16CXX series are:

RC (resistor capacitor)
LP (low power)
XT (crystal < 4 MHz)
HS (High speed)

The four types of oscillator available in the PIC17CXX series are:

RC (resistor capacitor)
LF (low power)
XT (crystal < 4 MHz)
EC (External Clock)

The four circuits are shown in Figure 3. This unique arrangement gives the designer the ability to optimize the performance of the PIC16/17 in terms of clock speed, type of resonator, and power consumption.

The RC Oscillator

The RC oscillator is a relaxation type similar to the popular 555 timer. The OSC1 pin is the input to a schmitt trigger.

The LP oscillator

The LP, or low power oscillator, is designed to trade speed for low power operation. Although this circuit shares the same topology (schematic) as the XT oscillator, the transistors used in the LP oscillator have a higher Rdss value and draw considerably less current. This configuration is optimum for the low frequency operation, because it trades the away unnecessary high frequency response for dramatically reduced operating currents.

The XT oscillator

The XT oscillator is designed to give a compromise between high frequency performance and modest power consumption. The gain of this oscillator is as much as 15 times higher than the LP oscillator. This middle range will be used for frequencies up to 4 MHz.

The HS oscillator

The HS oscillator is designed to give the maximum gain and frequency response. The current consumption is accordingly higher. The gain is roughly five times higher than that of the XT oscillator. This gives the PIC16/17 the ability to operate at frequencies up to 20 MHz.

5

© 1994 Microchip Technology Inc.

FIGURE 8: CRYSTAL DESIGN SHEET

XYZ INC.

DRAWN _____ DATE _____

APPROVED _____ DATE _____

FREQUENCY @ LOAD _____

MAKE TOLERANCE _____

MODE OF VIBRATION _____

SERIES RESISTANCE _____

MOTIONAL CAPACITANCE _____

STATIC CAPACITANCE _____

DRIVE LEVEL _____

SPURIOUS _____

ACTIVITY DIPS _____

AGEING _____

CRYSTAL DESIGN and TEST

OPERATING TEMPERATURE RANGE _____

Frequency Deviation _____

_ From Turn to Turn

_ Over Operating Temperature range

PACKAGE _____

TYPE OF SEAL _____

EVIRONMENTAL:

VIBRATION _____

SHOCK _____

CRYSTAL TEST

TEMPERATURE _____

THERMAL SHOCK _____

AGEING _____

GROSS LEAK _____

FINE LEAK _____

NOTES

© 1994 Microchip Technology Inc.

FIGURE 9: EXAMPLE CRYSTAL DESIGN SHEET

XYZ INC. **CRYSTAL DESIGN and TEST**

DRAWN _____ DATE _____

APPROVED _____ DATE _____

FREQUENCY @ LOAD 10MHz @ 32pF

MAKE TOLERANCE ± 20 PPM

MODE OF VIBRATION Fundamental

SERIES RESISTANCE 25 Ω Max.

MOTIONAL CAPACITANCE .028 pF ± 20%

STATIC CAPACITANCE _____

DRIVE LEVEL 1 mW Max.

SPURIOUS < -3dB

ACTIVITY DIPS < 3 PPM

AGEING < 2 PPM / Year

OPERATING TEMPERATURE RANGE -20 to +70°C

Frequency Deviation _____

✓ From Turn to Turn

_ Over Operating Temperature range

PACKAGE H/C 49

TYPE OF SEAL Resistance

EVIRONMENTAL: N/A

VIBRATION _____

SHOCK _____

CRYSTAL TEST

TEMPERATURE _____

THERMAL SHOCK _____

AGEING 10 Days at 85°C

GROSS LEAK 100%

FINE LEAK 13% AQL

NOTES

5

© 1994 Microchip Technology Inc.

APPENDIX A

The curves of Figure 5 are calculated using a general form developed by Bechmand in 1955. For any temperature T, a factional deviation from the frequency at the reference temperature T_0, is given in the form:

$$\frac{\Delta F}{F} = a(T-T_0) + b(T-T_0)^2 + (T-T_0)^3$$

where:

a	=	$-5.15 \times 10^{-6} * °(\theta - \theta_0)$
b	=	0.39×10^{-9} $-4.7 \times 10^{-9} * °(\theta - \theta_0)$
c	=	109.5×10^{-12} $-2 \times 10^{-12} * °(\theta - \theta_0)$
$(\theta - \theta_0)$	=	the difference between the intended angle and the zero temperature coefficient angle, in degrees of arc.

T_0 is the reference temperature and is usually taken as 25°C. The zero temperature coefficient angle is approximately -35.25° relative to the Y-axis. The exact angle which produces a zero temperature coefficient and the exact inflection temperature are both strongly dependent on several factors, including overtone and resonator geometry. A degree as a unit of angle is too coarse for sufficient resolution. The following coefficients are divided by 60 for units of minutes of arc.

REFERENCES AND BIBLIOGRAPHY (HIGHLY RECOMMEDED READING)

Ferking, M.E., *Crystal Oscillator Design and Temperature Compensation*, Van Norstrand Reinhold, New York, 1978.

Parzen, B., *Design of Crystal and other Harmonic Oscillators*, John Wiley and Sons, New York, 1983.

Salt, D., *The HY-Q Handbook of Quartz Crystal Devices*, Van Norstrand Reinhold, CO. Ltd, Berkshire, England.

This application note has been developed by:

Kim Peck
29 Wachusett Road
Needham, MA 02192
(617) 444-7748

© 1994 Microchip Technology Inc.

MICROCHIP

AN584

PICMASTER™ Support of Microsoft® Windows™ DDE

The PICMASTER system supports Windows Dynamic Data Exchange (DDE). This feature allows the contents of the trace buffer to be transferred to other windows applications such as Microsoft Excel™. This feature is invaluable to control systems designers who would like to plot real-time data to debug and fine tune an application. This application note will show how to set this up and graph system data.

THE TRACE BUFFER

The PICMASTER contains a 8K x 40 bit trace buffer. The fields within this buffer are broken up into three categories:

(1)	Current Address of instruction	16-Bits	**(ADDRESS)**
(2)	Data/Opcode Field	16-Bits	**(DATA)**
(3)	External Logic Analyzer inputs	8-Bits	**(EXT)**
		40-Bits	

Any instruction can be optionally traced or not traced; the trace for each instruction is enabled by the "T" field on the far left column of the program memory dump window. To set up the trace buffer, do the following:

1. Select the ***SETUP->TRACE SETTINGS*** window option to set up a range of addresses to be traced. These ranges are selecting specific memory addresses that will be traced if these instructions are executed. Individual instructions can also be trace enabled/disabled with the ***SHIFT+RIGHT_MOUSE*** button
2. Open the trace window by selecting the ***WATCH->TRACE MEMORY*** menu selection.
3. Click the ***RUN*** button after opening the Run box with the ***RUN->RUN BOX*** menu selection.
4. The processor will now run. At a later time you may hit the ***HALT*** button from the Run Box.
5. You will see the trace buffer fill with instructions and data if those instructions were executed and trace enabled. If you do not see any instructions in the trace buffer, check the Trace Settings in (1) and look for loops or deadlock situations that would prevent your program from executing these instructions.

Note: PICMASTER must be running and the trace buffer open with data displaying in order to use DDE for other Windows applications (such as Microsoft Excel as described in the next section).

SETTING UP EXCEL

After starting Excel with a blank spread sheet, select 100 rows in the first column by pressing the left mouse button and holding it down to drag across all cells. Next Type the following string in the box:

=PICMASTR|'c:\path\test.hex'!'data 0 99'

and hit ***CNTRL+SHIFT+ENTER***

The format for this command is as follows:

=PICMASTR|'***<hex_file>***'!'***<string>*** ***<start_address>*** ***<end_address>***'

- ***<hex_file>*** represents the full path and filename of the object code file that you have downloaded to program memory. This string must match what is displayed in the PICMASTER window handle at the top of the PICMASTER window.
- ***<string>*** defines the group of values that you want to look at in the trace buffer. The following are the three groups of trace information:
- **ADDRESS** for opcode address (16-bits Real-time data)
- **DATA** for the opcode coding itself (16-bits Real-time data)
- **EXT** for the external logic probes (8-bits Real-time data)

EXAMPLE 1: TRACE BUFFER DATA

Trace Buffer Address		Program Memory Address	Instruction Opcode (12,14,16 bit)	Address Label	Instruction	Comments
0	:	711	425		bcf porta,0x1	;Toggle Clock
1	:	712	525	BIT3	bsf porta,0x1	
2	:	713	305		rrf porta,W	;Clock in data
3	:	714	36F		rlf 0xF	

© 1994 Microchip Technology Inc.

- **RAM** for the current state of any file registers (8-bits, ***Not*** Real-time)
- **PROG** The current state of program memory (16-bits, ***Not*** Real-time)
- ***<start_address>*** specifies starting address location in the trace buffer to be transferred
- ***<end_address>*** specifies the ending address location in the trace buffer.

Note: Generally, the ***<end_address>*** must be greater than the ***<start_address>*** and the difference equal to the number of cells selected in Excel.

Hit ***SHIFT-CTL-ENTER*** and the data/opcode field values show up in the excel cells. You can now create hex values or strip off the literal encoding with formulas written for Excel.

MOVING LIVE DATA INTO THE TRACE BUFFER

Many applications would like to show trends in register values over time as the application runs real-time. For example, if the microcontroller software is using a file register **SPEED** to show the current shaft speed of a motor that the controller was controlling, the designer may want to capture the motor speed VS time when the controller is running in real-time. This can be accomplished with three different methods, two of which work with the PIC16CXX and PIC16C5X family, and all three work with the PIC17C42.

Method 1: Lookup Table (PIC16C5X, PIC16CXX, PIC17C42)

At the end of your code, add a look-up table that returns the literal value that is passed to it. This value can then be traced in the trace buffer through the DATA field and the data value stripped from the opcode. The lookup table that is added would look like the following:

```
    .                       ; Here is your code that calculates SPEED
    .
    MOVF   SPEED,W          ; Place the current value of the SPEED register into W
    CALL   TRACE_REG        ; Call TRACE_REG so the opcode is stored in trace buffer
    .
    .
    ORG (600-3)             ; This address must have bit 9 as a zero in 16C5X
    TRACE_REG               ; and must always have the ADDWF PC on address XX00h
            BTFSC   STATUS,Z ; Skip to Decrement if not zero
.T..        RETLW   0       ; if zero, return zero
            DECF    SPEED,W ; Compensate for table offset
            ADDWF   PC      ; Jump into lookup table and return same value as W-1
.T..        RETLW   1       ; Return 1
.T..        RETLW   2       ; Return 2
.T..        RETLW   3       ; Return 3
.T..        .               ; There are 250 RETURN instructions here -
.T..        .               ; RETLW 4 through RETLW 0FDh
.T..        RETLW   0FEh    ; Return FE
.T..        RETLW   0FFh    ; Return FF
```

Note: Since the **RETLW** instruction takes two cycles and the next instruction is prefetched, the trace buffer will contain two sequential values for every value that is traced with this method. The user can strip this intervening value out with the spread sheet.

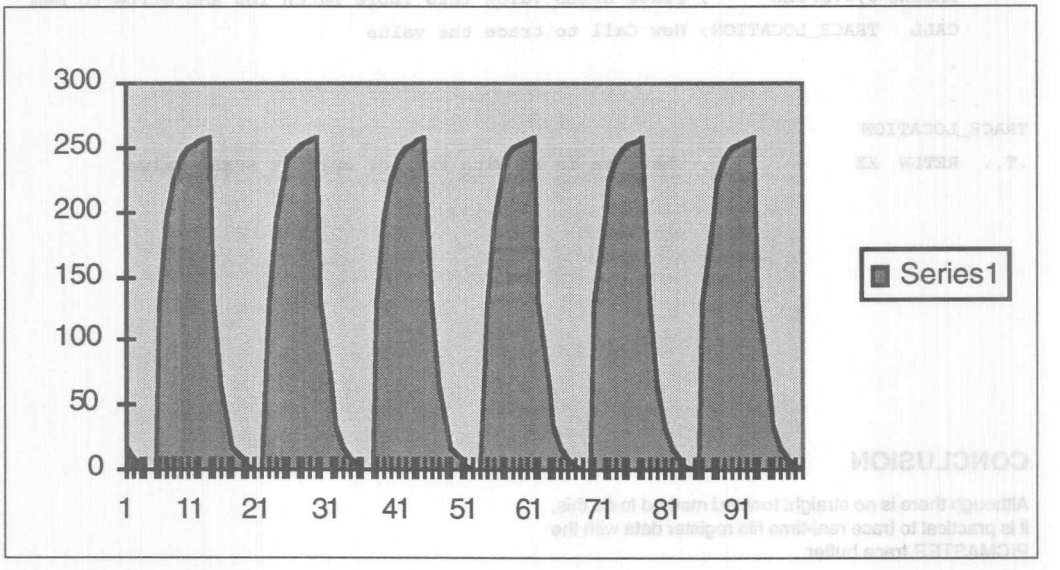
© 1994 Microchip Technology Inc.

Method 2: Using the Logic Analyzer external inputs (PIC16C5X, PIC16CXX, PIC17C42).

Connect the eight logic analyzer inputs TRC<7:0> to a currently unused digital output port of the PIC16/17. Set the port direction of this port to OUTPUT and when you want to look at a value, simply move the value to the port and set the trace bit on the second instruction following the move instruction. The following example shows this:

```
    .                   ; Code Initialization
    CLRW                ; Place 0 into W register
    OPTION   Port_B     ; Make all Port B pins outputs to feed variable state data to logic inputs
    .                   ; Use MOVWF DDRB instruction in place of OPTION for 17C42
    .
    .                   ; Place where Speed variable is modified and needs to be traced
    MOVF     SPEED,W    ; Place Speed value into W register
    MOVWF    Port_B     ; Place value into Port B
    .                   ; Execute normal instruction
.T..        .           ; Trace second instruction after Port B move to capture current value
```

FIGURE 1: EXCEL SPREADSHEET EXAMPLE OF REAL-TIME PSEUDO-SINE WAVE GENERATOR



5

© 1994 Microchip Technology Inc.

Method 3: Using TBLWRT instruction to modify program memory (PIC17C42 only)

The PIC17C42 supports table lookup and writing features that are not supported on the PIC16CXX family. The emulator supports write operations to program memory using the TBLWRT instruction. To capture real-time RAM data, the user simply sets up the table pointers to an unused portion of program memory space and loads the current register value into the table latch. Next the user issues the TBLWRT instructiono load the data into the least significant byte of the word used for trace. A simple call to this location with the trace bit set will capture the data. The following example shows this:

```
      MOVLW  TRACE_HIGH     ; PlaceTRACE_LOCATION MSB into W
      MOVWF  TBLPTRH        ; Move TRACE_LOCATION MSB into Table pointer HIGH
      MOVLW  TRACE_LOW      ; Place TRACT_LOCATION LSB into W
      MOVWF  TBLPTRL        ; Move TRACE_LOCATION LSB into Table pointer LOW
      TLWT   1,0B6h         ; Place RETLW opcode MSB into Table Latch High
      TABLWT 0,0,SPEED      ; Place SPEED value into Table Latch low and write to mem
      CALL   TRACE_LOCATION; Now Call to trace the value
      .
      .
TRACE_LOCATION
.T..  RETLW  XX             ; The data in XX with reflect current SPEED value
```

ANALYSIS OF VARIOUS METHODS

Method 1 requires an additional 259 words of code space at the end of memory but does not require any external port pin connections. Method 2 requires an 8-bit port connection to the external logic input pins but does not require any additional code space. Additionally, Method 2 does not have an intervening sequential address that occurs from the two cycle RETLW instruction used in Method 1 and Method 3.

Method 3 does not require the additional code of Method 1 or the port connections of Method 2, but this is only supported on the PIC17C42.

CONCLUSION

Although there is no straight forward method to do this, it is practical to trace real-time file register data with the PICMASTER trace buffer.

Author: John Day, Field Applications Engineer, Northeast Region (North America)

© 1994 Microchip Technology Inc.

SECTION 6
ASSP PRODUCTS APPLICATION NOTES

Calibrating the MTA11200 System - AN570 ..... 6- 1
Communicating with EEPROM in MTA8XXXX - AN571 ..... 6- 21
Hardware and Software Resolution for a Pointing Device - AN569 ..... 6- 33

6

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

MICROCHIP

AN570

TRUEGAUGE™ Calibrating the MTA11200 System

INTRODUCTION

This application note analyzes the calibration algorithm from theoretical and numerical approaches. It includes two calibration procedures, including information on setting up lab equipment, board modifications and additional hardware for the TrueGauge calibration. Lastly, it includes the methodology for re-biasing the comparators for different voltage, current and temperature ranges.

This calibration procedure can be modified to perform calibration in an automated manufacturing/test environment.

This procedure assumes the user is familiar with the TrueGauge demo kit P/N DV114001, documentation, hardware, and TrueGauge assemblies. Also that the TrueGauge demo software programs "TG Demo" and "Calibration" have been installed in Windows™ and the hardware has been functionally tested.

Required Equipment:

1. TrueGauge demo software 2.0 or higher. TG Demo and Calibration programs
2. TrueGauge demo kit P/N DV114001, or equivalent assemblies
3. DMM 4 1/2 digit (a second DMM is preferred)
4. Power Supply 14V, 1A minimum
5. Precise resistance source such as a Decade Resistance Box, Temperature Calibration Block (described in this procedure) or user designed resistance network
6. Assorted test leads, banana and clip styles

References:

1. MTA11200 Intelligent Battery Management IC Data Sheet DS40104
2. TrueGauge MTA11200 Introduction Kit User's Guide DS40108

The TrueGauge name and logo are trademarks of Microchip Technology, Inc.

Windows is a registered trademark of Microsoft Corporation.

Notes and Precautions:

1. This calibration procedure requires slight modification of the TrueGauge and Charger/Discharger board assemblies to simulate ideal battery and thermistor operation. The modifications should only be made by qualified personnel. Individual assemblies should not be interconnected during modification or prior to beginning the calibration procedure.
2. **When a power supply is used in place of a battery, the charger/discharger board must be kept in the discharge mode. If not, damage may occur to the power supplies and/or the charger/discharger board.**
3. The calibration accuracy is determined by tolerances determined by the users allowable limits with consideration of the test equipment tolerances, and deviation from the optimum values given.
4. The Temperature Calibration Block shown in the Appendix, is used in this description when performing thermistor calibration.
5. To enhance familiarization with the TrueGauge system it is recommended that the manual calibration be done before attempting the automated calibration.
6. Please read this application note entirely, from beginning to end, **before** attempting to calibrate a TrueGauge.

THE CALIBRATION SOFTWARE

The calibration source code is provided on the diskette in the TrueGauge demo kit P/N DV114001 and is also available on the Microchip BBS. Instructions for logging on to the BBS can be found in the *Microchip Data Book*, or the *Embedded Control Handbook*.

The TrueGauge installation procedure installs the Windows software and the calibration program which provides a more automated method of calibrating the modules. The calibration source code is written in QBASIC.

The manual calibration is performed using the TrueGauge demo Windows software. For purposes of explanation of the calibration process, the TrueGauge demo software is used to record the voltage, current and temperature readings corresponding to the known values. These known values were taken from the calibration source code, and represent optimum values.

The calibration program software is invoked by double-clicking on the "Calibration" icon in the TrueGauge program group. Upon doing so, a full DOS screen will appear. The user works in the bottom half of the screen to set up the appropriate calibration parameters.

© 1994 Microchip Technology Inc.

THEORY OF OPERATION

TrueGauge digitally integrates battery charge and discharge current to provide an accurate state of charge indication. The state of charge indicator depends on relative accuracy from one discharge cycle to the next. However, there are several parameters such as End of Discharge Voltage (EODV), and Maximum Fast Charge Temperature (MAXTFC) that are referenced to absolute voltage and temperature limits. TrueGauge's absolute accuracy is achieved via calibration of the comparators and their respective external components.

Each of the four comparators on the TrueGauge module uses a timed ramp for A/D conversion. Ideally, the ramps would have a slope of one and an offset, or y-intercept, of zero. However, because of component tolerances and drift, the ramp is not ideal. The calibration software uses the fundamental slope equation to calculate the calibration factors for each A/D.

$$y = mx+b$$

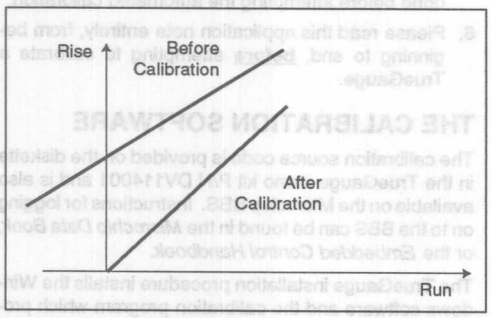
Slope "m" is first calculated by finding the difference between two points on the y-axis and dividing that value by the difference of two points on the x-axis.

$$m = (y_2-y_1)/(x_2-x_1)$$

Offset "b" is then calculated by rearranging the fundamental slope equation, where "y" is the known upper value and "x" is the corresponding value read and reported by TrueGauge.

$$b = y-mx$$

FIGURE 1 - SLOPE DIAGRAMS



The fundamental slope equation is expanded for voltage, current and temperature in the next section. Notice that the slope and y-intercept values are all scaled by 256_{10} or 0100_{16} or with an implied decimal point. This decimal point makes 256 equal to a slope of one. The hex values are stored in the EEPROM and used by the TrueGauge. However, the values are scaled by 256 in the demo software that runs in the Windows environment.

THE CALIBRATION HARDWARE

Charger/ Discharger Board

For a diagram of modifications to the Charger/Discharger Board, refer to drawing, "Charger/Discharger Board Calibration Modifications," in the Appendix section. To measure current, remove the resistor R7, 0.1 ohm from the charger/discharger board. R7 is physically located between the red and black banana jack connectors provided for ammeter connection. Retain the resistor for reinstallation after calibration.

Temperature Calibration Block

To perform thermistor calibration, it is recommended that the Temperature Calibration Block be constructed. Refer to the drawing, "Charger/Discharger Board Calibration Modifications". Before connecting the calibration block to the TrueGauge module the calibration resistance values must be set. The accuracy of the thermistor calibration is determined by the deviation from optimum values when adjusting the calibration block resistance's.

Adjusting the Calibration Resistance Values

Set the DMM to measure resistance and connect the test leads to the Temperature Calibration Block, at the TG T2 and the TG LD-/T1 connections. Select the resistance value to be adjusted using S1, and adjust R2 or R4 the 10 turn rheostat for 207807 ohms or 51058 ohms respectively.

TrueGauge Module

For a diagram of modifications to the TrueGauge Module for calibration, refer to drawing, "Diagram for TrueGauge Calibration," in the Appendix section. If the TrueGauge module has a battery attached, it must be disconnected and the power supply (V1) connected in its place. The power supply should initially be set to 0.0VDC, and connected to TrueGauge module. Connect the Positive lead to terminal LD+ and the negative lead to terminal LD-/T1, at the TrueGauge module or the calibration block.

MANUAL CALIBRATION OF THE TRUEGAUGE MODULE

This calibration procedure is performed using the TrueGauge demo software and is intended for familiarization of the concepts involved in calibrating a TrueGauge. This procedure does not calibrate the REFVAL parameter. As a precaution retain all initial calibration factors by updating them from the TrueGauge module and storing them to a hard disk in an *.h8m* file.

© 1994 Microchip Technology Inc.

Calibration Setup

Review the manual calibration procedure and refer to the diagram for TrueGauge calibration setup drawing in the Appendix.

Connect the charger/discharger board to the +6.0VDC power supply and the RS232 cable between the charger/discharger board and the host computer.

Connect the TrueGauge module IDC connector and modular plug to the charger/discharger board, and verify that the mode switch on the charger/discharger board is in the discharge position.

Set the power supply to 0.0VDC, and connect the TrueGauge Module LD+ and LD-/T1 to power supply (V1) + and - then LD-/T1 and T2 to the Temperature Calibration Block.

Connect the ammeter to the banana jacks on the charger/discharger board. Note that because R7 has been removed, the current reading in the run window will be 0mA if the ammeter is not connected.

Set the power supply V1 voltage to +6.0VDC and start the TrueGauge demo program. Use setup and select the appropriate COM port, upload the EEPROM data from the TrueGauge module then select parameters and move to the advanced factors screen (refer to the User's Guide).

Calibration Procedure

Consider the following manual calibration as an example, which was performed on an uncalibrated TrueGauge module. The values uploaded or these example values may be used. When the values in the demo program advanced factors screen are changed, the EEPROM data will need to be unlocked and the new values downloaded to the TrueGauge module before they effect the displayed readings in the run mode window.

The uncalibrated slope and offset values displayed in the advanced factors screen of the demonstration software were as follows: (Note: slopes = 1 and offsets = 0).

ISC:	Current Slope	=	256	(1.0)
VOC:	Voltage Offset	=	0	(0.0)
VSC:	Voltage Slope	=	256	(1.0)
TOC:	Temp. Offset	=	0	(0.0)
TSC:	Temp. Slope	=	256	(1.0)
REFVAL:	A/D Refer.	=	32767	

The data table which corresponds to the Windows values is shown below. From the TrueGauge data sheet, the calibration value addresses are shown in Table 2.

As explained above, a slope value of 256 in this window is equivalent to a slope of one. This slope of one and y-intercept of zero are ideal values.

Using these addresses and the data in Table 1, the correspondence between the EEPROM data and the Windows values can be verified. This table may be viewed by using Windows Notepad. Invoke Notepad and open *Filename.h8m*.

6

TABLE 1 - BREAK-OUT OF .H8M FILE CALIBRATION DATA

byte count	address	record type	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	check sum
10	0030	00	FF	7F	00	01	00	00	00	01	00	01	00	00	00	00	E0	2E	71
			REFVAL		VSC		VOC		ISC		TSC		TOC						

TABLE 2 - CALIBRATION PARAMETER ADDRESSES

ISC_{hh}	37_{16}	VOC_{hh}	35_{16}	VSC_{hh}	33_{16}	TOC_{hh}	$3B_{16}$	TSC_{hh}	39_{16}	$REFVAL_{hh}$	31_{16}
ISC_{lh}	36_{16}	VOC_{lh}	34_{16}	VSC_{lh}	32_{16}	TOC_{lh}	$3A_{16}$	TSC_{lh}	38_{16}	$REFVAL_{lh}$	30_{16}

© 1994 Microchip Technology Inc.

TABLE 3 - .H8M FILE PRIOR TO CALIBRATION

byte count	address	record type	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	check sum
10	0000	00	01	10	DB	AA	25	00	00	00	28	0A	28	0A	00	2F	00	00	B2
10	0010	00	5F	00	00	00	00	00	1A	10	0E	16	05	05	03	01	00	00	45
10	0020	00	04	00	01	01	00	00	00	00	00	10	80	00	49	A5	04	00	78
10	0030	00	FF	7F	00	01	00	00	00	01	00	01	00	00	00	00	E0	2E	71
10	0040	00	9A	DA	F3	FB	FD	FD	FD	FD	FD	FD	FD	FD	F1	CA	80	43	38
10	0050	00	FF	FF	FF	FF	FF	FF	FF	F5	EB	E1	D7	CC	C2	B8	AE	A3	D8
10	0060	00	06	00	0B	00	13	00	21	00	3B	00	68	00	B9	00	48	01	16
10	0070	00	42	02	03	04	1B	07	95	0C	48	16	74	27	DE	45	B8	76	A6
00	0000	01	FF																

Once the advanced factors values are selected, enter the Run mode, after each setting read and record the resulting values that the TrueGauge Module reports via the Windows demo software.

Disconnect the ammeter from the charger/discharger board. Note that because R7 has been removed, the Current reading in the Run window will be 0mA if the ammeter is not connected. Set the DMM to read DC volts.

Set the power supply to +6.0VDC and record the voltage reported by the TrueGauge as v_1'. Refer to the flowchart in the Appendix.

Next set the power supply to +14.0VDC and record the voltage reported by the TrueGauge as v_2'. Return the power supply V1 voltage to +6.0VDC, and disconnect the DMM.

Reset the DMM to read current and reconnect the ammeter to the banana jacks on the charger/discharger board. Note that the current reading in the Run window will indicate some value when the ammeter is connected.

Adjust the discharge potentiometer on the charger/discharger board, set the current to 0mA and -900mA reading on the ammeter and record the values for current reported by the TrueGauge as i_2' and i_1', respectively. (Note: a typical current reading in the discharge mode with current set for minimum is -15mA, if this occurs set both charger/discharger potentiometers to minimum then switch the mode switch to charge and adjust for 0mA reading on the ammeter. Return to the discharge mode as soon as an accurate reading is recorded).

Simulate the thermistor by selecting the resistor network using the Temperature Calibration Block S1, to known values of 51058 ohms and 207807 ohms for 40°C and 10°C respectively. Read the resulting values that the TrueGauge reports via the Windows demo software and record as t_1' and t_2'.

The values from the readings noted are applied to mathematical formulas. These formulas are extracted directly from the calibration source code. These calculations become the new slope and offset values.

VOLTAGE: VSC AND VOC

Note: The known upper and lower ramp voltage values were found in the "Calibrate Voltage" function of the calibration source code. The corresponding voltage values as reported by the TrueGauge module were read from the display screen of the Windows demo software.

v_2 = Applied known upper ramp voltage = 14.00 volts

v_1 = Applied known lower ramp voltage = 6.00 volts

v_2' = Reported voltage reading when 14.00 volts is applied

v_1' = Reported voltage reading when 6.00 volts is applied

m = $((v_2-v_1)/(v_2'-v_1'))*256$

m = ((14000mV-6000mV)/(14972m6391mV))*256

m = ((8000mV)/(8581mV))*256

VSC = m_v = 238.67 = 239

b = $v_2-(m*v_2')/256$

b = 14000mV-(238.67*14972mV)/256

b = 14000mV-13958.28mV

VOC = b_v = 41.72 = 42

© 1994 Microchip Technology Inc.

CURRENT : ISC

Note: The known upper and lower ramp current values were found in the "Calibrate Current" function of the calibration source code. The corresponding current values were read from the display screen of the Windows demo software.

i_2 = Applied known upper ramp current = -900mA

i_1 = Applied known lower ramp current = 0mA

i_2' = Reported current reading corresponding to i_2 = -1088mA

i_1' = Reported current reading corresponding to i_1 = 0mA

m = $((i_2-i_1)/(i_2'-i_1'))*256$

m = ((-900mA-0mA)/(-1088mA-0mA))*256

m = ((-900mA)/(-1088mA))*256

ISC = m_i = 211.76 = 212

TEMPERATURE: TSC AND TOC

Note: The known upper and lower ramp temperature/ resistance values were found in the "Calibrate Temperature" function of the calibration source code. The two resistance values were used in place of the thermistor to simulate the thermistor at 10°C and 40°C. The corresponding temperature values were read from the display screen of the Windows demo software.

t_2 = Applied known upper ramp temperature = 10°C

t_1 = Applied known lower ramp temperature = 40°C

t_2' = Reported temperature reading corresponding to t_2 = 182°C-256°C = -74°C

t_1' = Reported temperature reading corresponding to t_1 = 218°C-256°C = -38°C

m = $((t_2-t_1)/(t_2'-t_1'))*256$

m = ((10°C-40°C)/(-74°C-(-38°C)))*256

m = ((-30°C)/(-36°C))*256

TSC = m_t = 213.33 = 213

b = $((t_2-((m/256)*t_2'))*256$

b = (10°C-((.833)*(-74°C)))*256

b = (10°C-(-61.6°C))*256

TOC = b_t = 18329.6 = 18330

The newly calibrated slope and offset values should then be input in the advanced factors screen (refer to the User's Guide) of the demonstration software running in Windows.

ISC:	Current Slope	=	212
VOC:	Voltage Offset	=	42
VSC:	Voltage Slope	=	239
TOC:	Temp. Offset	=	18330
TSC:	Temp. Slope	=	213
REFVAL:	A/D Refer.	=	32589

Upon performing a download with these new calibration factors to the EEPROM, the values would be reflected in the data table (Table 2).

6

TABLE 4 - .H8M FILE AFTER CALIBRATION AND DOWNLOADING TO EEPROM

byte count	address	record type	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	check sum
10	0000	00	01	10	DB	AA	25	00	00	00	28	0A	28	0A	00	2F	00	00	B2
10	0010	00	5F	00	00	00	00	00	1A	10	0E	16	05	05	03	01	00	00	45
10	0020	00	04	00	01	01	00	00	00	00	00	10	80	00	49	A5	04	00	78
10	0030	00	4D	7F	EF	00	2A	00	D4	00	D5	00	9A	47	00	00	E0	2E	71
10	0040	00	9A	DA	F3	FB	FD	FD	FD	FD	FD	FD	FD	FD	F1	CA	80	43	38
10	0050	00	FF	FF	FF	FF	FF	FF	FF	F5	EB	E1	D7	CC	C2	B8	AE	A3	D8
10	0060	00	06	00	0B	00	13	00	21	00	3B	00	68	00	B9	00	48	01	16
10	0070	00	42	02	03	04	1B	07	95	0C	48	16	74	27	DE	45	B8	76	A6
00	0000	01	FF																

Verify the TrueGauge readings in the run mode using the method described earlier. Confirm that the voltage, current, and temperature readings are within the users tolerances. If needed, the slope and offset calculations may be calculated.

When calibration is complete, use the Lock EEPROM command to protect the values, then exit out of the demo program. The test equipment, TrueGauge module and charger/discharger board may be de-energized and disconnected. Reconnect the battery and a thermistor to the TrueGauge module, reinstall R7 into the charger/ discharger board.

© 1994 Microchip Technology Inc.

THE AUTOMATED CALIBRATION

The calibration exercise in the previous section was performed manually using the TrueGauge demo Windows software. However, the installation procedure also installs a calibration program which provides a more automated method of calibrating the modules.

Calibration Setup

Review this procedure and refer to the diagrams in the Appendix.

Connect the charger/discharger board to the wall power supply, and the RS232 cable between the charger/discharger board and the host computer.

Connect the TrueGauge module IDC connector and modular plug to the charger/discharger board, and verify that the mode switch on the charger/discharger board is in the discharge position.

Set the power supply to 0.0VDC, and connect the TrueGauge module LD+ and LD-/T1 to power supply (V1) + and - then LD-/T1 and T2 to the Temperature Calibration Block.

Connect the ammeter to the banana jacks on the charger/discharger board. Note that because R7 has been removed, the current reading in the run window will be 0mA if the ammeter is not connected.

Set the power supply V1 voltage to +6.0VDC and start the calibration program.

The calibration software is invoked by double-clicking on the "Calibration" icon in the TrueGauge program group. Upon doing so, a full DOS screen will appear. A reproduction of the bottom halves of the screens appear below. The user works in the bottom half of the screen to set up the appropriate calibration parameters.

Setup Begin Set NP Voltage Current Temp End Restore Quit

Initialize Calibration Options

1. Open calibration result log file (optional).
2. Select comm port.
3. Open EEPROM data file (optional).
4. Specify load for current calibration (optional).

<Enter=Initialize> <Tab=skip to Next> <Back Tab=skip to Previous>

Initialization is started when the user presses the <ENTER> key.

Upon pressing <ENTER>, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit

Open a calibration result log file.

<Enter=Open> <Tab=skip to Next> <Back Tab=skip to Previous>

Press the <ENTER> key again, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit

Open a calibration result log file.

Enter results log filename:

<Enter=Open> <Tab=skip to Next> <Back Tab=skip to Previous>

© 1994 Microchip Technology Inc.

In this screen, the software is prompting the user to set up a file to store the results of calibration. This file stores the part serial number and the calibration values in a single file which is appended with new values from each separate calibration sequence.

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Enter COM port number (1, 2, 3 or 4).

<1, 2, 3 or 4=Port Number>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

In this screen, the calibration software is asking to be referred to the COM port which the charger/discharger board is attached to. This is so that communication may be established with the TrueGauge module.

TYPE : The COM port number to which the TrueGauge is attached. Upon doing so, the screen will then read:

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Open an EEPROM initialization data file.

<Enter=Open>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

Press the <ENTER> key again, the screen will read:

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Open an EEPROM initialization data file.
Enter EEPROM DATA filename:

<Enter=Open>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

In this screen, the software is prompting the user to specify a default file for TrueGauge module(s) to be calibrated.

TYPE: *Filename* (this is an optional filename) If filename is not specified the screen will display an alarm message.

Press the <ENTER> key the screen will read:

(If the alarm message occurs press <ENTER> a second time).

This default file will initialize the entire EEPROM with the data contained in the file. The A/D calibration factors contained in this file will set all of the A/D gain values to one and offsets to zero, thus facilitating the simple mathematical formulas shown in the Manual Calibration section in this document.

© 1994 Microchip Technology Inc.

TYPE : *Filename.h8m*

Press the <ENTER> key.

If the file does not exist the screen will display an alarm message.

Press the <ENTER> key again, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit
Open an EEPROM initialization data file.
<Enter=Open> <Tab=skip to Next> <Back Tab=skip to Previous>

Press the <TAB> key, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit
Change reference load (1000 milliamps) for Current Calibration.
<Enter=Change> <Tab=skip to Next> <Back Tab=skip to Previous>

Press the <ENTER> key again, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit
Change reference load (1000 milliamps) for Current Calibration.
Enter new load in milliamps:
<Enter=Change> <Tab=skip to Next> <Back Tab=skip to Previous>

If the power supply used in the lab setup is incapable of outputting -1000 mA of current, the reference load should be changed.

If <TAB> is pressed, the reference load will default to -1000 mA, and the software will jump ahead to the "start calibration" screen. Otherwise, the user can lower this value to one that the power supply can deliver.

If <ENTER> is pressed, the screen will read:

Setup Begin Set NP Voltage Current Temp End Restore Quit
Enter new load in milliamps:
<Enter=Open> <Tab=skip to Next> <Back Tab=skip to Previous>

© 1994 Microchip Technology Inc.

TYPE: the new value as a positive number, the program will change this to a negative value, to indicate *discharge* current.

Press: <ENTER>. The software will then proceed to the next screen.

```
Setup  Begin  Set NP  Voltage  Current  Temp  End  Restore  Quit

Start Calibration Sequence

    1. Specify part number.
    2. Verify broadcast mode is off.
    3. Verify comm link to TrueGauge is operational.
    4. Load current calibration factors.

<Enter=Start>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

In this screen, the software is waiting for the user to begin calibration, which starts when the user presses the <ENTER> key. Upon pressing the <ENTER> key, the screen reads:

```
Setup  Begin  Set NP  Voltage  Current  Temp  End  Restore  Quit

Enter part/serial number:
```

The user must enter a number to identify the module being calibrated.

The length may vary from one to eight characters.

After entering the number,

Press: <ENTER>. The screen will begin a routine which will display this information:

```
Setup  Begin  Set NP  Voltage  Current  Temp  End  Restore  Quit

Checking for broadcast enabled. Broadcasting is disabled.
Unlocking EEPROM. EEPROM Unlocked.
Reading battery data.
Uploading original calibration factors: [**********]
```

In this screen, the software is informing the user of the status for different operations which are being performed in preparation for calibration. The software will always disable broadcast. If it is not disabled, the software will proceed to disable it. EEPROM unlocking will then take place so that new calibration factors can be written to the EEPROM. Finally, the software reads and uploads the original calibration values which are currently in the EEPROM. Upon completed uploading of these calibration factors, the software will proceed to the next screen.

© 1994 Microchip Technology Inc.

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Set Normalization Point.

<Enter=Set NP>      <Tab=skip to Next>      <Back Tab=skip to Previous>
```

In this screen, the software is prompting the user to set the normalization point. This is an A/D reference point, and is typically set once, at the beginning of a calibration. It is reflected as the numerical value stored in REFVAL of the Windows demo software, and is a representation of the time value for REFC input to TrueGauge. Once set, the normalization point compensates for any changes in TrueGauge module components over time and temperature. During normal operation, the present measured value for REFC is compared with the stored value in the EEPROM, and the ISENC, BATVC and TEMPC TrueGauge inputs are each compensated by the difference in the measured versus stored REFC values. Press : The <ENTER> key, the screen reads:

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Setting normalization point. One moment, please.
Resetting TrueGauge. One moment, please.
Reset successful.
Unlocking EEPROM. EEPROM Unlocked.
```

The normalization point should be set immediately before calibrating. Calibration should not be done without first setting the normalization point. This is because the REFC input is used to compensate for changes in components over time and temperature. If new calibration values are being offset by a REFC delta which was established at a different time in the life of the TrueGauge module, optimal accuracy will not be achieved. The software will then proceed to the next screen.

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Calibrate voltage.

<Enter=Calibrate>      <Tab=skip to Next>      <Back Tab=skip to Previous>
```

In this screen, the software is prompting the user to begin calibrating the voltage parameters. Upon pressing the <ENTER> key, the screen reads:

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Initializing correction factors. One moment, please: [****]
Set voltage to 6.000 volts then hit any key.
```


© 1994 Microchip Technology Inc.

In this screen, the software is prompting the user to adjust the power supply to +6.0VDC, which is the lower end of the voltage slope defined by the formulas in the program. Upon setting the power supply to six volts and pressing any key, the screen reads:

Setup Begin Set NP **Voltage** Current Temp End Restore Quit

Set voltage to 14.000 volts then hit any key.

Upon setting the power supply to +14.0VDC, which is the upper end of the voltage slope defined by the formulas in the program, and pressing any key. The software takes the inputs and their corresponding reading applies slope and intercept formulas to the numbers, deriving new calibration factors. The screen then reads:

Setup Begin Set NP **Voltage** Current Temp End Restore Quit

Storing new correction factors. One moment, please: [****]

Read battery data.

In this screen, the software is informing the user that the new voltage calibration parameters are being loaded to the EEPROM. The software will then proceed to the next screen.

Setup Begin Set NP Voltage **Current** Temp End Restore Quit

Calibrate Current.

<Enter=Calibrate> <Tab=skip to Next> <Back Tab=skip to Previous>

In this screen, the software is prompting the user to begin calibrating the current parameters. Upon pressing the <ENTER> key, the screen reads:

Setup Begin Set NP Voltage **Current** Temp End Restore Quit

Calibrate Current.

Initializing correction factors. One moment, please: [**]

Set current to 0mA then hit any key (Hit Esc to Quit).

6

© 1994 Microchip Technology Inc.

In this screen, the software is prompting the user to move the slide switch on the charger/discharger board to the "sleep" position for a moment to achieve a zero current reading. This switch must be returned to the discharge pin very quickly (before the next polling cycle).

If not, the software will report that it is unable to communicate with the TrueGauge, and calibration will have to be restarted. Upon doing so and pressing any key, the screen reads:

Setup	Begin	Set NP	Voltage	**Current**	Temp	End	Restore	Quit

Set current to - xxxx mA then hit any key.

Note: (xxxx = value entered earlier)

Upon setting the discharge potentiometer on the charger/discharger board to obtain the specified current (observed on the DMM) and pressing any key, the software takes the inputs and their corresponding reading and applies slope and intercept formulas to the numbers, deriving new calibration factors. The screen then reads:

Setup	Begin	Set NP	Voltage	**Current**	Temp	End	Restore	Quit

Storing new correction factors. One moment, please: [**]

Reading battery data.

In this screen, the software is informing the user that the new current calibration parameters are being loaded to the EEPROM. The software will then proceed to the next screen.

Setup	Begin	Set NP	Voltage	Current	**Temp**	End	Restore	Quit

Calibrate Temperature.

<Enter=Calibrate> <Tab=skip to Next> <Back Tab=skip to Previous>

© 1994 Microchip Technology Inc.

In this screen, the software is prompting the user to begin calibrating the temperature parameters.

Press the <ENTER> key , the screen reads:

Setup	Begin	Set NP	Voltage	Current	**Temp**	End	Restore	Quit

Initializing correction factors. One moment, please: [****]

Set thermistor to 51058 Ω then hit any key.

In this screen, the software is prompting the user to move the toggle switch shown on the Temperature Calibration Block to the position which selects the 51058 ohm resistor. Upon doing so and pressing any key, the screen reads:

Setup	Begin	Set NP	Voltage	Current	**Temp**	End	Restore	Quit

Initializing correction factors. One moment, please: [****]

Set thermistor to 51058 Ω then hit any key.

Set thermistor to 207807 Ω then hit any key.

The software is now prompting the user to move the toggle switch to the position which selects the 207807 ohm resistor. Upon doing so the software takes the inputs and their corresponding readings and applies slope and intercept formulas to the numbers, deriving new calibration factors. These two resistance measurements were taken from the specification for the thermistor which is included with the TrueGauge module in the development kit. 207807 ohms corresponds to a temperature of 10°C and 51058 ohms to a temperature of 40°C. Upon doing so the software takes the inputs and their corresponding readings and applies slope and intercept formulas to the numbers, deriving new calibration factors. The screen then reads:

Setup	Begin	Set NP	Voltage	Current	**Temp**	End	Restore	Quit

Storing new correction factors. One moment, please: [****]

Reading battery data.

© 1994 Microchip Technology Inc.

In this screen, the software is informing the user that the new temperature calibration parameters are being loaded to the EEPROM. The software will then proceed to the next screen.

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

End Calibration.

<Enter=End Calibration>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

In this screen, the software is prompting the user to end calibration. Pressing the <Enter> key will end calibration, The screen then changes to:

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

End Calibration.
Locking EEPROM.

<Enter=End Calibration>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

This screen completes storage of the new calibration parameters to the TrueGauge EEPROM. The screen then changes to the "Start Calibration Sequence" screen.

```
Setup   Begin   Set NP   Voltage   Current   Temp   End   Restore   Quit

Start Calibration Sequence
    1. Specify part number.
    2. Verify broadcast mode is off.
    3. Verify comm link to TrueGauge is operational.
    4. Load current calibration factors.

<Enter=Start>   <Tab=skip to Next>   <Back Tab=skip to Previous>
```

Press the <TAB> key until the Quit option is highlighted and appears in the window then press the <ENTER> key. This ends this procedure.

Verify the TrueGauge readings in the run mode using the method described earlier. Confirm that the voltage, current, and temperature readings are within the users tolerances. If needed, the slope and offset calculations may be calculated.

When the actual readings are within tolerance, use the Lock EEPROM command to protect the values, then exit out of the demo program. The test equipment, TrueGauge Module and charger/discharger board may be de-energized and disconnected. Reconnect the battery and a thermistor to the TrueGauge module, reinstall R7 into the charger/discharger board.

© 1994 Microchip Technology Inc.

COMPARATOR BIASING FOR UNIQUE A/D RANGING

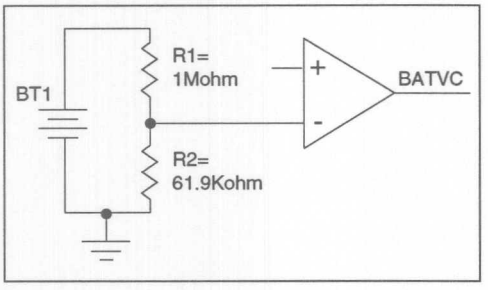
Minimum and maximum voltage biases configured in the present design at the comparator inputs are provided in Table 5. Individual explanations of these bias voltages will follow. Input common-mode voltage for the comparator used on the TrueGauge module is:

V_{DD}-1.5volts=3.5volts

This common mode voltage information was used to determine the bias voltage ranges for the comparators as shown in Table 5.

The TrueGauge module schematic included in the data sheet has been biased for a certain parameter range. Individual comparator biasing can be adjusted to accommodate special cases.

FIGURE 2 - VOLTAGE COMPARATOR SCHEMATIC



In Figure 2, the voltage divider of R1 and R2 was selected for a highest allowable battery voltage of 64 volts.

$Voltage_{max}$ = (61.9KΩ/(1MΩ+61.9KΩ))*64 volts = 3.7 volts

$Voltage_{min}$ = (61.9KΩ/(1MΩ+61.9KΩ))*2.7 volts = .16 volts

3.7 volts is on the high end of the linear comparator input curve. If a higher battery voltage is desired, this voltage divider must be recalculated to maintain proper comparator biasing.

In Figure 3, the bias voltage is calculated as shown below:

$Voltage_{bias}$ = (572KΩ/(1MΩ+572KΩ))*5 volts = 1.8 volts

The .1ohm current sense resistor and battery can be modeled as a voltage source, such as the one shown in Figure 3. As current flows into the positive terminal of the battery during charge, a "c (charging)" voltage is developed across R6. Alternately, as current flows out of the battery during discharge, a "d (discharging)" voltage is developed. To maintain minimal power loss, the voltage drop across R6 was designed to equal plus/minus .5 volts. By applying the same voltage divider to this voltage as was applied to the SWVCC voltage, and applying Kirchoff's Voltage Law, it is shown that the input pin to the comparator is kept between 1.62 to 1.98 volts. This configuration facilitates 1mA A/D resolution.

By applying Ohm's law:

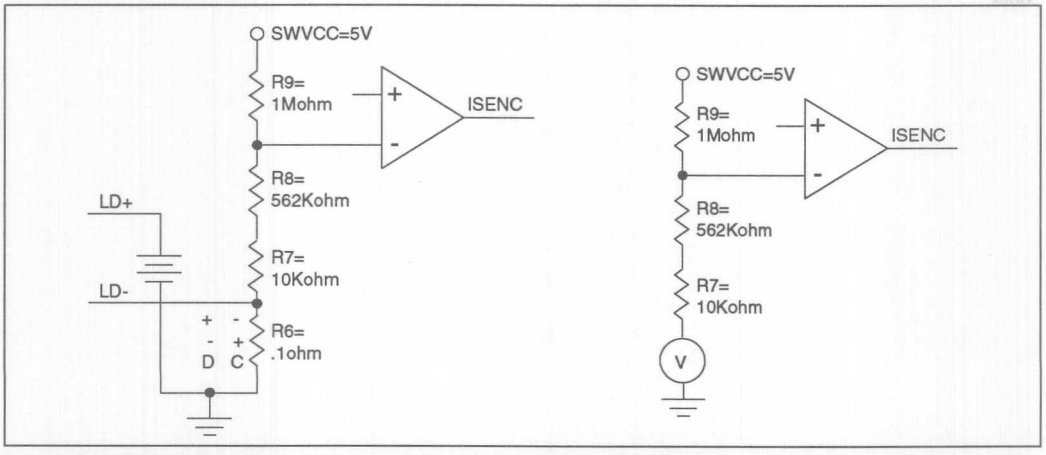
I = .5volts/.1Ω = 5A

6

TABLE 5 - BIAS LIMITS FOR COMPARATORS IN DEVELOPMENT KIT CONFIGURATION

	Voltage Comparator	Current Comparator	Temperature Comparator
Voltage $_{Max}$	3.7 Volts	1.98 Volts	1.5 Volts
Voltage $_{Min}$	.16 Volts	1.62 Volts	.022 Volts

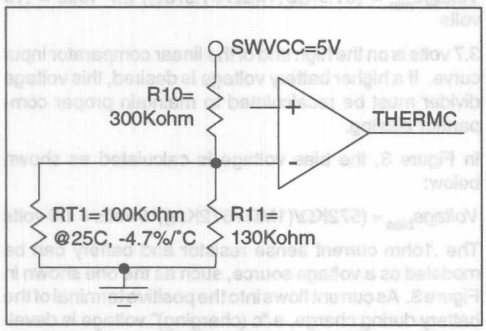
FIGURE 3 - CURRENT COMPARATOR SCHEMATIC



© 1994 Microchip Technology Inc.

Therefore, to boost the current range for the TrueGauge, the current sense resistance must be reduced. For example, a current range of 10 amps could be achieved by setting R6 = .05ohm. However, this will only allow 2mA A/D resolution.

FIGURE 4 - TEMPERATURE COMPARATOR SCHEMATIC



Refer to Figure 4 for the following explanation.

Per the thermistor specification, the highest calculated temperature/resistance values are:

-55°C 12,162,871Ω

150°C 1358Ω

Calculating the equivalent parallel resistance of RT1 and R11 at these temperature limits, the following values were found:

$R_{-55°C}$ = 128,625Ω

$R_{150°C}$ = 1,344Ω

Finally, using these equivalent resistances, the upper and lower points on the input voltage curve can be calculated.

$Voltage_{max}$ = (128,625Ω/(300KΩ+128,625Ω))*5 volts = 1.5 volts

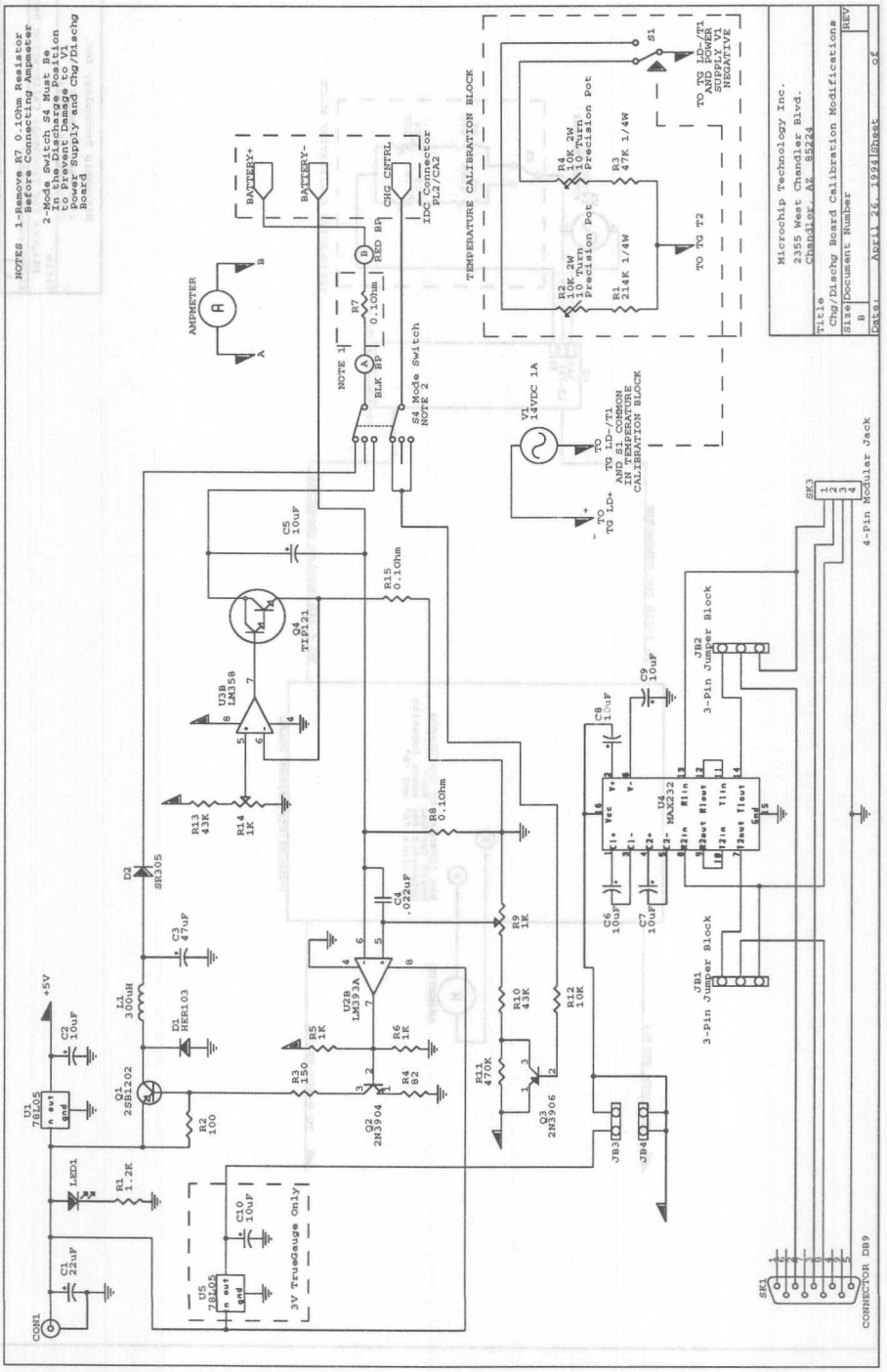
$Voltage_{min}$ = (1,344Ω/(300KΩ+1,344Ω))*5 volts = .022 volts

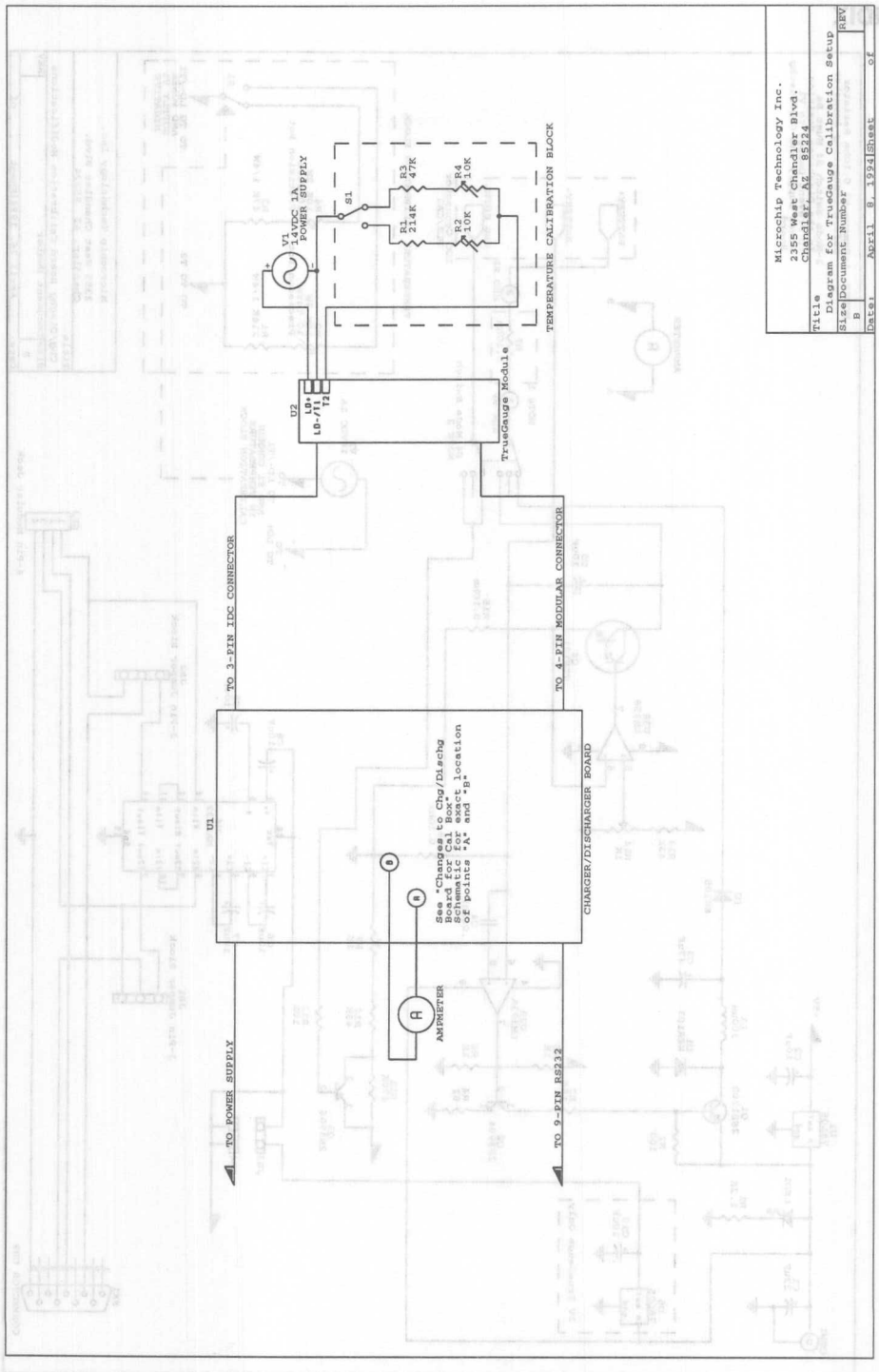
CONCLUSION

Calibration of the TrueGauge is very versatile. The operation is simple enough that it can be performed with a minimum amount of equipment in a lab environment. However it is also flexible enough to allow fully automated batch calibration in a manufacturing environment.

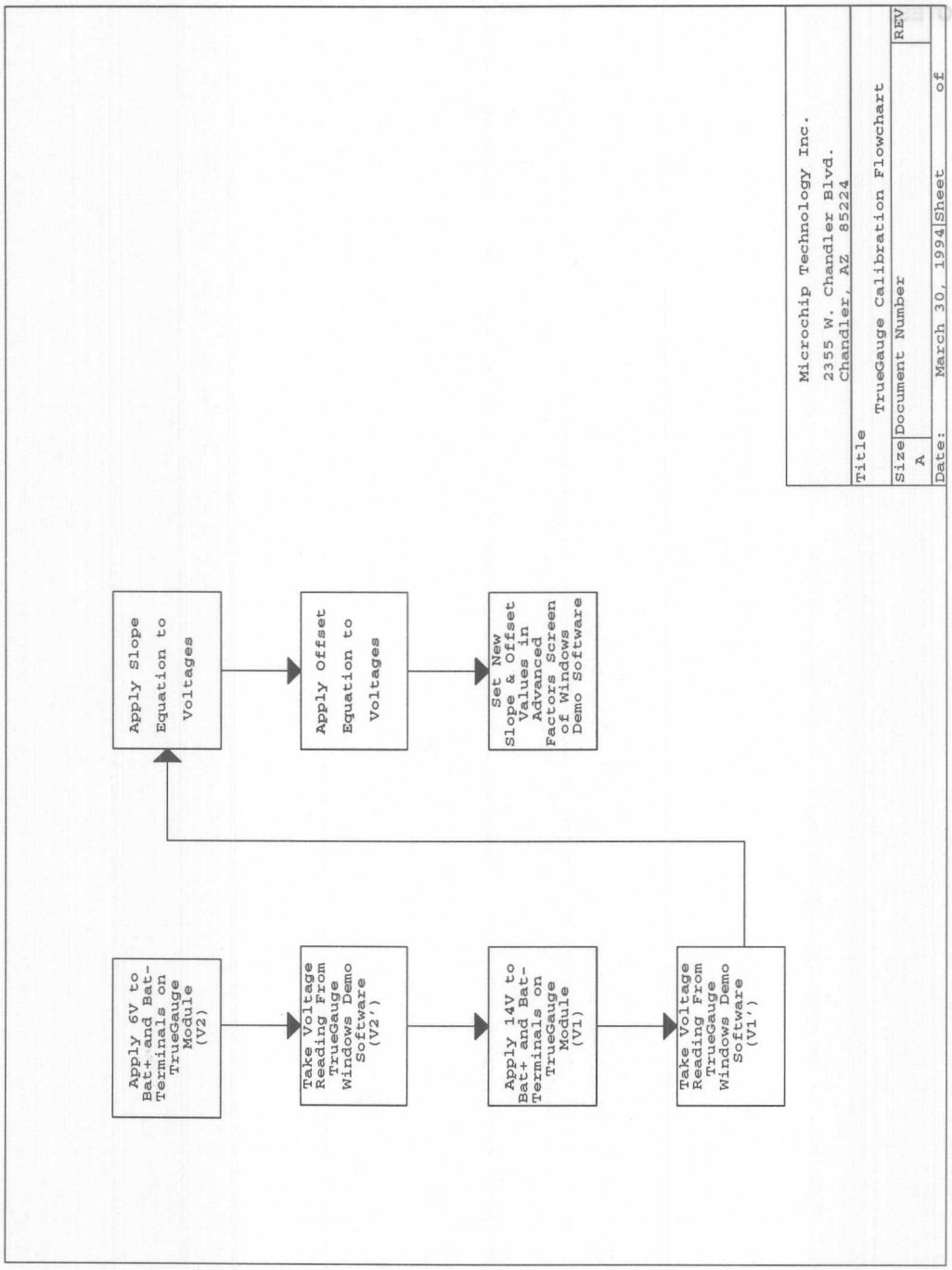
© 1994 Microchip Technology Inc.

APPENDIX


NOTES 1-Remove R7 0.1Ohm Resistor Before Connecting Ammeter
2-Mode Switch S4 Must Be In the Discharge Position To Prevent Damage To Power Supply and Chg/Dischg Board
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224
Chg/Dischg Board Calibration Modifications
Document Number
April 26, 1994
TEMPERATURE CALIBRATION BLOCK
AMMETER
BATTERY+
BATTERY-
CHG CNTRL
IDC Connector PL2/CX2
S4 Mode Switch NOTE 2
S1
R7 0.1Ohm
R15 0.1Ohm
R8 0.1Ohm
R4 10K 2W 10 Turn Precision Pot
R3 47K 1/4W
R2 10K 2W 10 Turn Precision Pot
R1 214K 1/4W
TO TG T2
TO TG LD-/T1 AND PROVIDE SUPPLY V1 NEGATIVE
V1 14VDC 1A
TG LD-/T1 AND S1 COMMON IN TEMPERATURE CALIBRATION BLOCK
TO TG LD+
Q4 TIP121
U3B LM358
R13 43K
R14 1K
D2 SR305
C5 10uF
C4 .022uF
C3 47uF
L1 300uH
D1 HER103
U2B LM393A
R5 1K
R6 1K
R3 150
R4 82
Q2 2N3904
Q1 2SB1202
R2 100
R9 1K
R10 43K
R11 470K
Q3 2N3906
R12 10K
U1 78L05
C2 10uF
LED1
R1 1.2K
C1 22uF
CON1
U5 78L05
C10 10uF
3V TrueGauge Only
C8 10uF
C9 10uF
C6 10uF
C7 10uF
U4 MAX232
JB2 3-Pin Jumper Block
JB1 3-Pin Jumper Block
JB3
JB4
SK3
4-Pin Modular Jack
SK1
CONNECTOR DB9



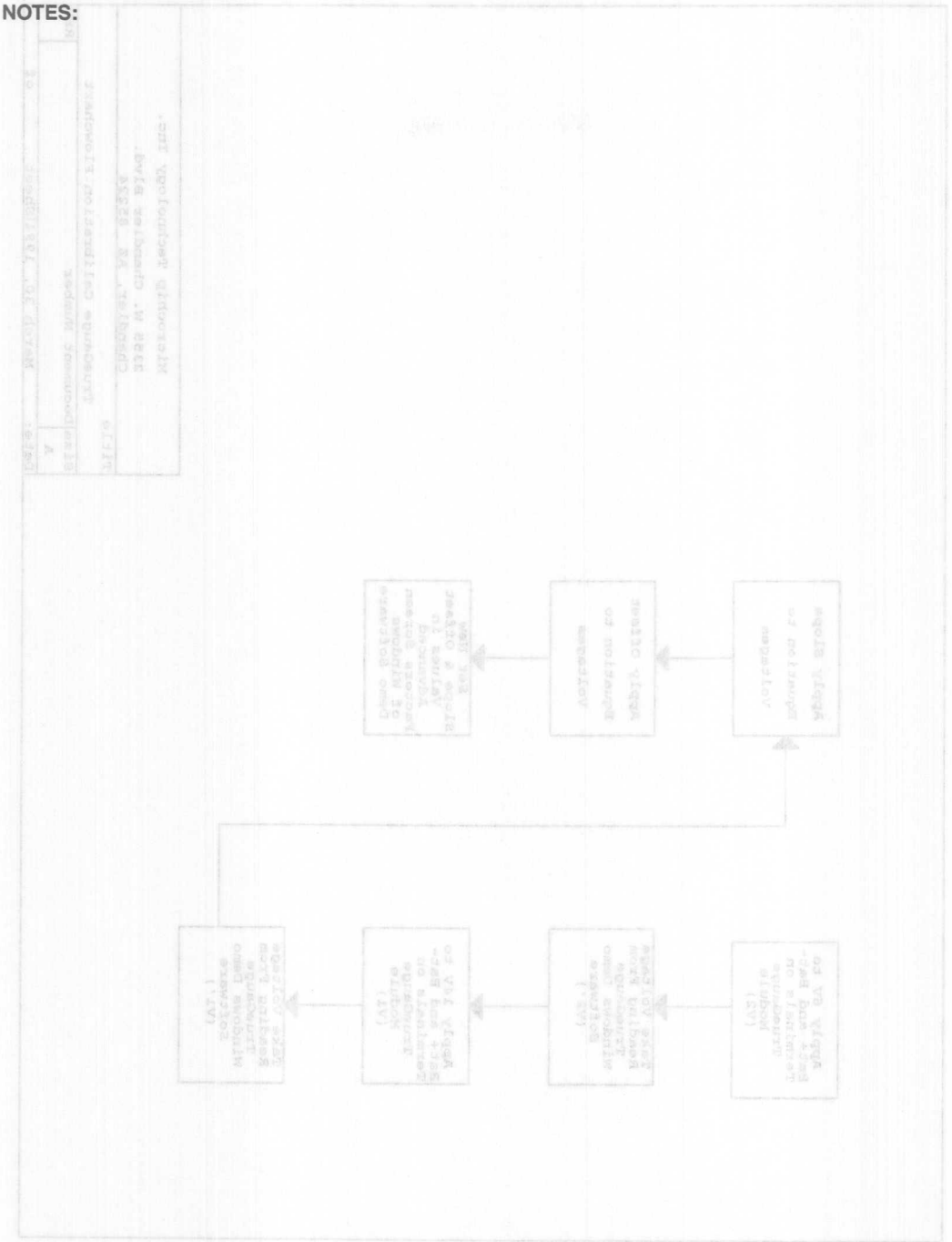
© 1994 Microchip Technology Inc.



6

© 1994 Microchip Technology Inc.

NOTES:



© 1994 Microchip Technology Inc.

Communicating With EEPROM In MTA85XXX

1.0 INTRODUCTION

The Microchip MTA85XXX family of microcontrollers are multichip modules which contain a PIC16C54A or PIC16C58A microcontroller and a Microchip Technology 24LC01B or 24LC02B EEPROM. This application note is intended to provide users with the most efficient code for I²C™-like communication between the EEPROM and the microcontroller in a PICSEE™ configuration, which will leave the user a maximum amount of code space and data RAM for the core application. The code contained in this application note will allow the PIC16C5XA microcontroller to act as the I²C master. The 24LC0XB EEPROM devices have I²C slave protocol implemented on-chip. Please refer to the Microchip Databook for further details on the 24LC0XB EEPROM. Note that this application note is optimized for use in either the PICSEE, or PIC16C5X plus EEPROM from Microchip. If a discrete PIC16C5X and generic serial EEPROM are to be used, refer to application note AN554 which details a complete I²C implementation.

The files which compose this application note (and their encapsulated post script print files) can be found on the Microchip BBS. The EEPROM communication code contained within this application note can be easily inserted into the users application code.

The frequency of operation is assumed to be 4MHz and the code has been verified over the entire operating voltage range of the MTA85XXX. This code was tested with the fuse settings:

OSC = XT, WDT = ON, CP = OFF, CKSUM = 6715

2.0 OPERATION

2.1 General Operation

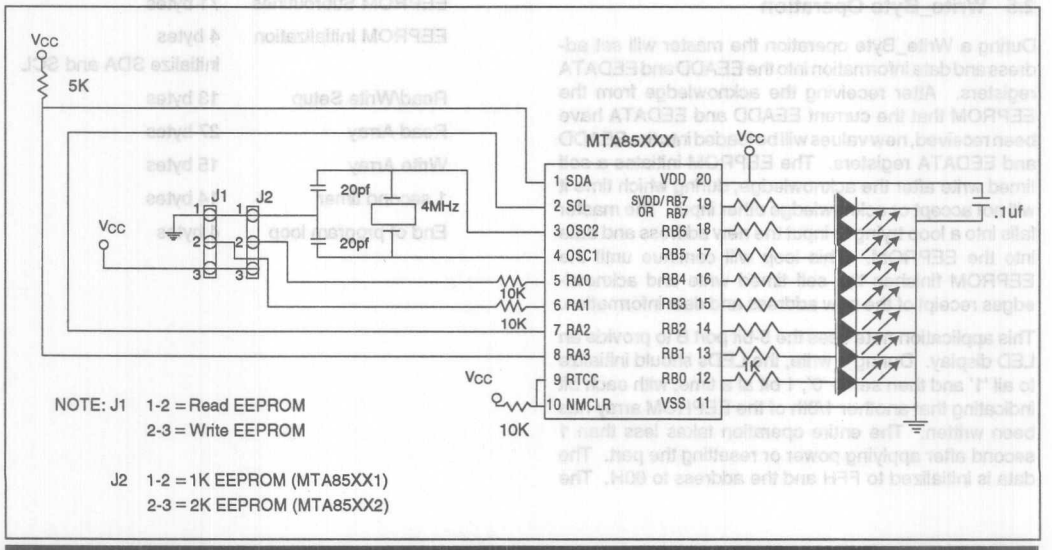
This application note will function with both bonding options of the MTA85XXX. Please refer to the MTA85XXX datasheet for further details. This application note will light LEDs on port B to indicate that the data is being written to the EEPROM or to output data read from the EEPROM.

The SDA and SCL bits of the EEPROM must be initialized to '1's before calling any of the EEPROM subroutines. This is done in the beginning of the MAIN routine. The start bit will not be initiated properly if these bits are not initialized prior to calling the EEPROM routines.

The registers EEADD and EEDATA are used to communicate EEPROM address and data information to and from the EEPROM, as described below.

6

FIGURE 2-1 SCHEMATIC



© 1994 Microchip Technology Inc.

2.2 Hardware Configuration

The schematic for this application note is located in figure 2-1. For the user's application, the pins used for SDA and SCL can be easily redefined and the switches will not be necessary. This application example shows the MTA85XXX driving an array of LEDs to indicate writing to the EEPROM is in progress.

2.3 Switches for array size and to determine read/write

Bits RA0 and RA1 are used to determine if the user would like to read or write the array and to select the array size (1K or 2K). Bits RA2 and RA3 are used for SCL and SDA.

RA0 = '0' for 1K EEPROM

RA1 = '0' for read EEPROM

RA0 = '1' for 2K EEPROM

RA1 = '1' for write EEPROM

2.4 EEPROM Subroutines

The code in this application note is capable of performing byte writes, current address reads, and random address reads. The names of the subroutines are:

Write_Byte	Write data supplied in register EEDATA at address supplied in register EEADD
Read_Random	Read data into register EEDATA from address supplied in register EEADD
Read_Current	Read data into register EEDATA from current address in EEPROM memory.

Note that the EEPROMs used in the MTA85XXX devices are also capable of page writes, however, page writes are not detailed by this application note. Refer to AN567 for details on page write.

2.5 Write_Byte Operation

During a Write_Byte operation the master will set address and data information into the EEADD and EEDATA registers. After receiving the acknowledge from the EEPROM that the current EEADD and EEDATA have been received, new values will be loaded into the EEADD and EEDATA registers. The EEPROM initiates a self timed write after the acknowledge, during which time it will not accept or acknowledge other input. The master falls into a loop trying to input the new address and data into the EEPROM. This loop will continue until the EEPROM finishes the self timed write and acknowledges receipt of the new address and data information.

This application note uses the 8-bit port B to provide an LED display. During a write, the LEDs should initialize to all '1' and then set to '0', 1 bit at a time, with each bit indicating that another 1/8th of the EEPROM array has been written. The entire operation takes less than 1 second after applying power or resetting the part. The data is initialized to FFH and the address to 00H. The address is incremented by 1 and the data decremented by 1 after each byte is written, until the EE array is full. An AAH is output to port B after the write has completed (actually as the last byte is writing).

2.6 Read_Current and Read_Random Operation

A read operation will attempt to read the data at the supplied or current address until an acknowledge is received. If an acknowledge is not received, the part will time out after a WDT period. If a Read_Random is being performed, the address is supplied in EEADD. If a Read_Current is being performed, the current address pointer inside the EEPROM is used. In both Read_Random and Read_Current the data is read into the EEDATA register.

In this application note, the port B LEDs are used to display the array content during a read operation. A 1-second timer is called between each byte so the data can be read on the LEDs. Note that with the MTA85X1X version (RB7 = SEEVDD), RB7 must always = '1' when the EEPROM is being accessed. The EEPROM address pointer (used for a read current address) will be reset to the last EEPROM address if RB7 is set to a '0' at any time. Due to this, the RB7 pin is always set to a '1' during a Read_Current operation (even if the data read is a '0' in the high order bit, a one will be output on RB7 to keep from resetting the EEPROM's internal address). MTA85X0X users may want to comment out the line which sets the RB7 to a '1' during a Read_Current operation, so that they will receive the data read from the EEPROM on the LEDs without the data appearing on RB7 altered.

2.7 Code Size

The total amount of program memory space required is 148 bytes. The major sections are as follows:

EEPROM Subroutines	71 bytes
EEPROM Initialization	4 bytes initialize SDA and SCL
Read/Write Setup	13 bytes
Read Array	27 bytes
Write Array	15 bytes
1 second timer	14 bytes
End of program loop	4 bytes

© 1994 Microchip Technology Inc.

3.0 TIMINGS (ASSUME 4MHZ)

Detailed timing diagrams are given in Section 8.

Write operation will take approximately 350µs + standard write time (Figure 8-1).

Read random address operation will take approximately 350µs (Figure 8-2).

Read current address will take approximately 250µs (Figure 8-3).

4.0 I²C COMPATIBILITY

NOTE: Code is not strictly I²C compatible on all timing parameters, but is designed to work with a Microchip 24LC01B or 24LC02B EEPROM from 3 to 6.25 volts. Commented NOPs have been included which can be put in place to gain I²C timing compatibility. Also note that '1's are driven onto the bus (not floated) by the master in most write circumstances, except where bus contention from the EEPROM slave could result.

5.0 ASSEMBLY CODE LISTING

```
TITLE "APP-NOTE PER I2C-BUS"
LIST P=16C54 , C=132
;LIST   P=PIC16C58  C=132

; PIC16C5X to 24LC01B or 24LC02B EEPROM communication code.
; Based on Franco code.

; FUSE SETTINGS:
; OSC = XT, WDT = ON, CP = OFF, CKSUM =

; INTRODUCTION:
; The Microchip MTA85XXX family of microcontrollers are multichip modules
; which contain a PIC16C5XA microcontroller and a 24LC01B or 24LC02B EEPROM.
; This application note is intended to provide users with highly compressed
; assembly code for communication between the EEPROM and the Microcontroller,
; which will leave the user a maximum amount of code space for the core
; application.

; For use in a users application, the file EEPROM.asm has been generated.
; This file contains the EEPROM subroutines and assembler constants used in
; these subrourines, stripped from this application note.  Although the file
; cannot be compiled, it is available for easy insertion into the user's
; application code.

; EEPROM SUBROUTINES:
; Write_Byte    - Write data supplied in EEDATA at address supplied in EEADD
; Read_Random   - Read data into EEDATA from address supplied in EEADD
; Read_Current - Read data into EEDATA from address currently in EEPROM

; OPERATION:
; For detailed operating information and other important information about
; this application note, read file AN571.doc

;*****************************************************************************
;**************************  Variable Listing  ***************************
;*****************************************************************************
OK          EQU     01H
NO          EQU     00H
;INDIRECT    EQU     00H         ; Indirect Address Register
;RTCLOCK     EQU     01H         ; Real time counter clock
PC          EQU     02H         ; Program counter
STATUS      EQU     03H         ; Status register
;PD          EQU     03H         ; Power down bit in STATUS
PORTA       EQU     05H         ; Port A control register
PORTB       EQU     06H         ; Port B control register
I2C_PORT    EQU     05H         ; Port A control register, used for I2C
```

© 1994 Microchip Technology Inc.

```
SCL         EQU     02H         ; EEPROM Clock, SCL (I/O bit 2)
SDA         EQU     03H         ; EEPROM Data,  SDA (I/O bit 3)
EE_OK       EQU     07H         ; Bit 7 in PC_OFFSET used as OK flag for EE
PC_OFFSET   EQU     07H         ; PC offset register (low order 4 bits),
                                ;  value based on operating mode of EEPROM.
                                ;  Also, bit 7 used for EE_OK flag
EEADDR      EQU     08H         ; EEPROM Address
EEDATA      EQU     09H         ; EEPROM Data
EEBYTE      EQU     0AH         ; Byte sent to or received from
                                ; EEPROM (control, address, or data)
COUNTER     EQU     0BH         ; Bit counter for serial transfer


   ORG     0
   GOTO    START

;*************************************************************************
;************************  EEPROM Subroutines  ***************************
;*************************************************************************
; Communication for EEPROM based on I2C protocol, with Acknowledge.
;
; Byte_Write: Byte write routine
;       Inputs:  EEPROM Address   EEADDR
;                EEPROM Data      EEDATA
;       Outputs: Return 01 in W if OK, else return 00 in W
;
; Read_Current: Read EEPROM at address currently held by EE device.
;       Inputs:  NONE
;       Outputs: EEPROM Data       EEDATA
;                Return 01 in W if OK, else return 00 in W
;
; Read_Random: Read EEPROM byte at supplied address
;       Inputs:  EEPROM Address    EEADDR
;       Outputs: EEPROM Data       EEDATA
;                Return 01 in W if OK, else return 00 in W
;
; Note: EEPROM subroutines will set bit 7 in PC_OFFSET register if the
;       EEPROM acknowledged OK, else that bit will be cleared.  This bit
;       can be checked instead of referring to the value returned in W
;*************************************************************************

;********************* Set up EEPROM control bytes ***********************
;*************************************************************************
READ_CURRENT
    MOVLW   B'10000100'    ; PC offset for read current addr.  EE_OK bit7='1'
    MOVWF   PC_OFFSET      ; Load PC offset
    GOTO    INIT_READ_CONTROL

WRITE_BYTE
    MOVLW   B'10000000'    ; PC offset for write byte.  EE_OK: bit7 = '1'
    GOTO    INIT_WRITE_CONTROL

READ_RANDOM
    MOVLW   B'10000011'    ; PC offset for read random.  EE_OK: bit7 = '1'

INIT_WRITE_CONTROL
    MOVWF   PC_OFFSET      ; Load PC offset register, value preset in W
    MOVLW   B'10100000'    ; Control byte with write bit, bit 0 = '0'

START_BIT
     BCF     I2C_PORT,SDA  ; Start bit, SDA and SCL preset to '1'


;******* Set up output data (control, address, or data) and counter ********
;*************************************************************************
PREP_TRANSFER_BYTE
    MOVWF   EEBYTE         ; Byte to transfer to EEPROM already in W
    MOVLW   B'00000011'    ; SDA and SCL set to output
    TRIS    I2C_PORT
```


© 1994 Microchip Technology Inc.

```
    MOVLW   .8              ; Counter to transfer 8 bits
    MOVWF   COUNTER

;************  Clock out data (control, address, or data) byte  ************
;*****************************************************************************
OUTPUT_BYTE
    BCF     I2C_PORT,SCL  ; Set clock low during data set-up
    RLF     EEBYTE        ; Rotate left, high order bit into carry bit
    BCF     I2C_PORT,SDA  ; Set data low, if rotated carry bit is
    SKPNC                 ;   a '1', then:
    BSF     I2C_PORT,SDA  ; reset data pin to a one, otherwise leave low
    BSF     I2C_PORT,SCL  ; clock data into EEPROM
    DECFSZ  COUNTER       ; Repeat until entire byte is sent
    GOTO    OUTPUT_BYTE
;*************************  Acknowledge Check  *******************************
;*****************************************************************************
    MOVLW   B'00001011'     ; SDA = input, SCL = output
    SKPNC                   ; if SDA = 1 then tristate port to allow
    TRIS    I2C_PORT        ;   pullup to hold '1', avoiding bus contention
                    ;   if EEPROM acks in < 1us after clock goes low
    BCF     I2C_PORT,SCL    ; Set SCL low, 0.5us < ack valid < 3us
    TRIS    I2C_PORT        ; If SDA = '0' wait until SCL is low to set SDA to
                    ;   input. If done above, could have sent STOP bit
;   NOP                     ; May be necessary for SCL Tlow  at low voltage,
                    ;   also give resistor time to pull up bus if last
                    ;   bit written = '0' and there is no ack from slave
    BSF     I2C_PORT,SCL    ; Raise SCL, EEPROM acknowledge still valid
    BTFSC   I2C_PORT,SDA    ; Check SDA for acknowledge (low)
    BCF     PC_OFFSET,EE_OK ; If SDA not low (no ack), set error flag
    BCF     I2C_PORT,SCL    ; Lower SCL, EEPROM release bus
    BTFSS   PC_OFFSET,EE_OK ; If no error continue, else stop bit
    GOTO    STOP_BIT


;*****  Set up program counter offset, based on EEPROM operating mode  *****
;*****************************************************************************
    MOVF    PC_OFFSET,W
    ANDLW   B'00001111'
    ADDWF   PC
    GOTO    INIT_ADDRESS      ;PC offset=0, write control done, send address
    GOTO    INIT_WRITE_DATA   ;PC offset=1, write address done, send data
    GOTO    STOP_BIT          ;PC offset=2, write done, send stop bit
    GOTO    INIT_ADDRESS      ;PC offset=3, write control done, send address
    GOTO    INIT_READ_CONTROL ;PC offset=4, send read control
    GOTO    READ_BIT_COUNTER  ;PC offset=5, set counter and read byte
    GOTO    STOP_BIT          ;PC offset=6, random read done, send stop


;**********  Initialize EEPROM data (address, data, or control) bytes  ******
;*****************************************************************************
INIT_ADDRESS
    INCF    PC_OFFSET       ; Increment PC offset to 2 (write) or to 4 (read)
    MOVF    EEADDR,W        ; Put EEPROM address in W, ready to send to EEPROM
    GOTO    PREP_TRANSFER_BYTE

INIT_WRITE_DATA
    INCF    PC_OFFSET       ; Increment PC offset to go to STOP_BIT next
    MOVF    EEDATA,W        ; Put EEPROM data in W, ready to send to EEPROM
    GOTO    PREP_TRANSFER_BYTE

INIT_READ_CONTROL
    BSF     I2C_PORT,SCL    ; Raise SCL
    INCF    PC_OFFSET       ; Increment PC offset to go to READ_BIT_COUNTER next
    MOVLW   B'10100001'     ; Set up read control byte, ready to send to EEPROM
    GOTO    START_BIT       ; bit 0 = '1' for read operation
```

6

© 1994 Microchip Technology Inc.

```
;************************** Read EEPROM data ****************************
;************************************************************************
READ_BIT_COUNTER
    MOVLW   .8              ; Set counter so 8 bits will be read into EEDATA
    MOVWF   COUNTER

READ_BYTE
    BSF     I2C_PORT,SCL    ; Raise SCL, SDA valid.  SDA still input from ack
    SETC                    ; Assume bit to be read = 1
    BTFSS   I2C_PORT,SDA    ; Check if SDA = 1
    CLRC                    ; if SDA not = 1 then clear carry bit
    RLF     EEDATA          ; rotate carry bit (=SDA) into EEDATA;
    BCF     I2C_PORT,SCL    ; Lower SCL
    DECFSZ  COUNTER         ; Decrement counter
    GOTO    READ_BYTE       ; Read next bit if not finished reading byte


;****************** Generate a STOP bit and RETURN **********************
;************************************************************************
STOP_BIT
    BCF     I2C_PORT,SDA    ; SDA=0, on TRIS, to prepare for transition to '1'
    MOVLW   B'00000011'     ; SDA and SCL set to outputs, Bit0 and Bit1 ' input
    TRIS    I2C_PORT
    BSF     I2C_PORT,SCL    ; SCL = 1 to prepare for STOP bit
;   NOP                     ; 4 NOPs necessary for I2C spec Tsu:sto = 4.7us
    BSF     I2C_PORT,SDA    ; Stop bit, SDA transition to '1' while SCL high
    BTFSS   PC_OFFSET,EE_OK ; Check for error
    RETLW   NO              ; if error, send back NO
    RETLW   OK              ; if no error, send back OK

;   Note: SDA and SCL still being driven by master, both set to outputs.
;************************************************************************
;************************ End EEPROM Subroutines ************************


;************************************************************************
;***************************** MAIN routine *****************************
;************************************************************************
START

;************************** Initialize EEPROM ***************************
;************************************************************************
    MOVLW   B'00000011'     ; SDA and SCL = output, Bit0 and Bit1 = input
    TRIS    I2C_PORT
    BSF     I2C_PORT,SCL    ; Initialize SCL and SDA to '1'
    BSF     I2C_PORT,SDA

;*** VARIABLE FOR MAIN ROUTINE THAT ARE NOT USED IN EEPROM SUBROUTINES ****
;************************************************************************

COUNT       EQU     0CH ; REGISTER FOR # OF BYTES IN ARRAY WRITTEN OR READ
LEDS        EQU     0DH ; REGISTER TO HOLD DATA TO OUTPUT TO PORT
TIMER_OL    EQU     0EH ; REGISTER FOR OUTER  LOOP VARIABLE FOR TIMER
TIMER_ML    EQU     0FH ; REGISTER FOR MIDDLE LOOP VARIABLE FOR TIMER
TIMER_IL    EQU     10H ; REGISTER INNER  LOOP VARIABLE FOR TIMER
EE1KPARTIAL EQU     10H ; 1/8TH OF 1K ARRAY (128 BYTES TOTAL)
EE2KPARTIAL EQU     20H ; 1/8TH OF 2K ARRAY (256 BYTES TOTAL)
EE1KFULL    EQU     80H ; 128
EE2KFULL    EQU     00H ; WILL = FFH AFTER DECREMENT, EFFECTIVELY = 256
SIZE        EQU     00H ; BIT0 ON PORT WILL BE READ TO DETERMINE ARRAY SIZE
READORWRITE EQU     01H ; BIT1 ON PORT WILL BE READ TO DETERMINE IF USER
                        ; WANTS TO READ OR WRITE THE ARRAY
EEVDD       EQU     07H ; BIT FOR EEPROM VDD ON MTA85X1X DEVICES
```


© 1994 Microchip Technology Inc.

```
;*********************  Read / Write  Set-Up  ****************************
;*****************************************************************************

    MOVLW   000H
    TRIS    PORTB               ; PORTB = OUTPUT
    MOVWF   EEADDR              ; 1ST ADDRESS = 00H
    MOVLW   0FFH                ; INITIALIZE DATA TO 'FF'
    MOVWF   LEDS                ; INITIALIZE PORTB OUTPUT FOR % DONE INDICATOR
    MOVWF   PORTB               ; SET PORT B TO ALL '1's
    MOVWF   EEDATA              ; INITIALIZE DATA TO ALL '1'

    BTFSS   PORTA,READORWRITE ; IF RA1 = '1' THEN WRITE, RA1 = '0' THEN READ
    GOTO    SETSIZE_READ

SET_SIZE
    MOVLW   EE1KPARTIAL       ; INIT COUNTER TO 1/8 OF ARRAY SIZE, 1K
    BTFSC   PORTA,SIZE        ; IF RA0 = '0' THEN 1K EEPROM, IF '1' THEN 2K
    MOVLW   EE2KPARTIAL       ; INIT COUNTER TO 1/8 OF ARRAY SIZE, 2K
    MOVWF   COUNT

;*****************************  Write Array  *********************************
;*****************************************************************************
CALL_WRITE
    CALL    WRITE_BYTE
    CLRWDT                      ; DON'T TIME OUT DURING EE OPERATIONS
    BTFSS   PC_OFFSET,EE_OK     ; TEST EE_OK BIT DETERMINED DURING ACK CHECK
    GOTO    CALL_WRITE          ; IF NO ACKNOWLEDGE, ASSUME PART IS STILL
    INCF    EEADDR              ; WRITING AND KEEP TRYING UNTIL GET THROUGH.
    DECF    EEDATA              ; IF PART DOES ACK, CHANGE DATA AND ADDRESS
    DECFSZ  COUNT               ; EVERY 1/8TH OF ARRAY CHANGE OUTPUT LEDs
    GOTO    CALL_WRITE

;     *******  UPDATE LED OUTPUTS TO INDICATE ARRAY BEING WRITTEN *******
    BTFSS   LEDS,06H            ; IF BIT6 IS CLEAR BEFORE SHIFT THEN ARRAY IS
    GOTO    INFINITELOOP        ; FINISHED WRITING (EXCEPT LAST BYTE STILL NOT
                                ; DONE) DO NOT CLEAR BIT7 OR MTA85X1X PRODUCTS
                                ; WILL LOSE THEIR VDD AND NOT FINISH LAST BYTE
    BCF     STATUS,0            ; SET CARRY BIT TO '0'
    RLF     LEDS                ; INCREMENT % DONE INDICATOR
    MOVF    LEDS,W
    MOVWF   PORTB
    GOTO    SET_SIZE

;************************  Timer 1-second  *********************************
;*****************************************************************************
TIMER
    MOVLW   .107
    MOVWF   TIMER_OL
OUTTER_LOOP
    CLRWDT
    MOVLW   .25
    MOVWF   TIMER_ML
MIDDLE_LOOP
    MOVLW   .123
    MOVWF   TIMER_IL
INNER_LOOP
    DECFSZ  TIMER_IL
    GOTO    INNER_LOOP
    DECFSZ  TIMER_ML
    GOTO    MIDDLE_LOOP
    DECFSZ  TIMER_OL
    GOTO    OUTTER_LOOP
    RETLW   00H
;*****************************************************************************

;*****************************  Read Array  **********************************
```

6

© 1994 Microchip Technology Inc.

```
;*************************************************************************
SETSIZE_READ
    MOVLW   EE1KFULL          ; INIT COUNTER TO ARRAY SIZE, 1K
    BTFSC   PORTA,SIZE        ; IF RA0 = '0' THEN 1K EEPROM, IF '1' THEN 2K
    MOVLW   EE2KFULL          ; INIT COUNTER TO ARRAY SIZE, 2K
    MOVWF   COUNT

CALL_READ_RANDOM
    CALL    READ_RANDOM
    BTFSS   PC_OFFSET,EE_OK   ; TEST EE_OK BYTE DETERMINED DURING ACK CHECK
    GOTO    CALL_READ_RANDOM  ; IF NO ACKNOWLEDGE, TRY AGAIN UNTIL RESET.
    INCF    EEADDR
    MOVF    EEDATA,W
    MOVWF   PORTB
    CALL    TIMER             ; SLOW DOWN OUTPUT SO IT IS READABLE ON LEDs
    BSF     PORTB,EEVDD       ; Always set RB7 to '1' for MTA85X1X devices
    DECFSZ  COUNT             ;   May see flash on LED.  If rise time is
    GOTO    CALL_READ_RANDOM  ;   to slow, EEPROM may not be powered-up yet

; RESET SIZE_READ
    MOVLW   EE1KFULL          ; INIT COUNTER TO ARRAY SIZE, 1K
    BTFSC   PORTA,SIZE        ; IF RA0 = '0' THEN 1K EEPROM, IF '1' THEN 2K
    MOVLW   EE2KFULL          ; INIT COUNTER TO ARRAY SIZE, 2K
    MOVWF   COUNT

CALL_READ_CURRENT
    CALL    READ_CURRENT
    BTFSS   PC_OFFSET,EE_OK   ; TEST EE_OK BYTE DETERMINED DURING ACK CHECK
    GOTO    CALL_READ_CURRENT ; IF NO ACKNOWLEDGE, TRY AGAIN UNTIL RESET.
    BSF     EEDATA,EEVDD      ; Always set RB7 to '1' for MTA85X1X devices
    MOVF    EEDATA,W          ; During READ_CURRENT RB7 CANNOT go to a '0'
    MOVWF   PORTB             ; or EEPROM internal address will be reset
    CALL    TIMER             ; SLOW DOWN OUTPUT SO IT IS READABLE ON LEDs
    DECFSZ  COUNT
    GOTO    CALL_READ_CURRENT
;   NOTE: Data observed on port B will always show a '1' regardless of what
;         was read.  MTA85X1X pinout devices (RB7=SVDD) will reset the EEPROM
;         internal address pointer if RB7 = 0.  If the user has the MTA85X0X
;         pinout, the BSF EEDATA,EEVDD line above can be commented out.

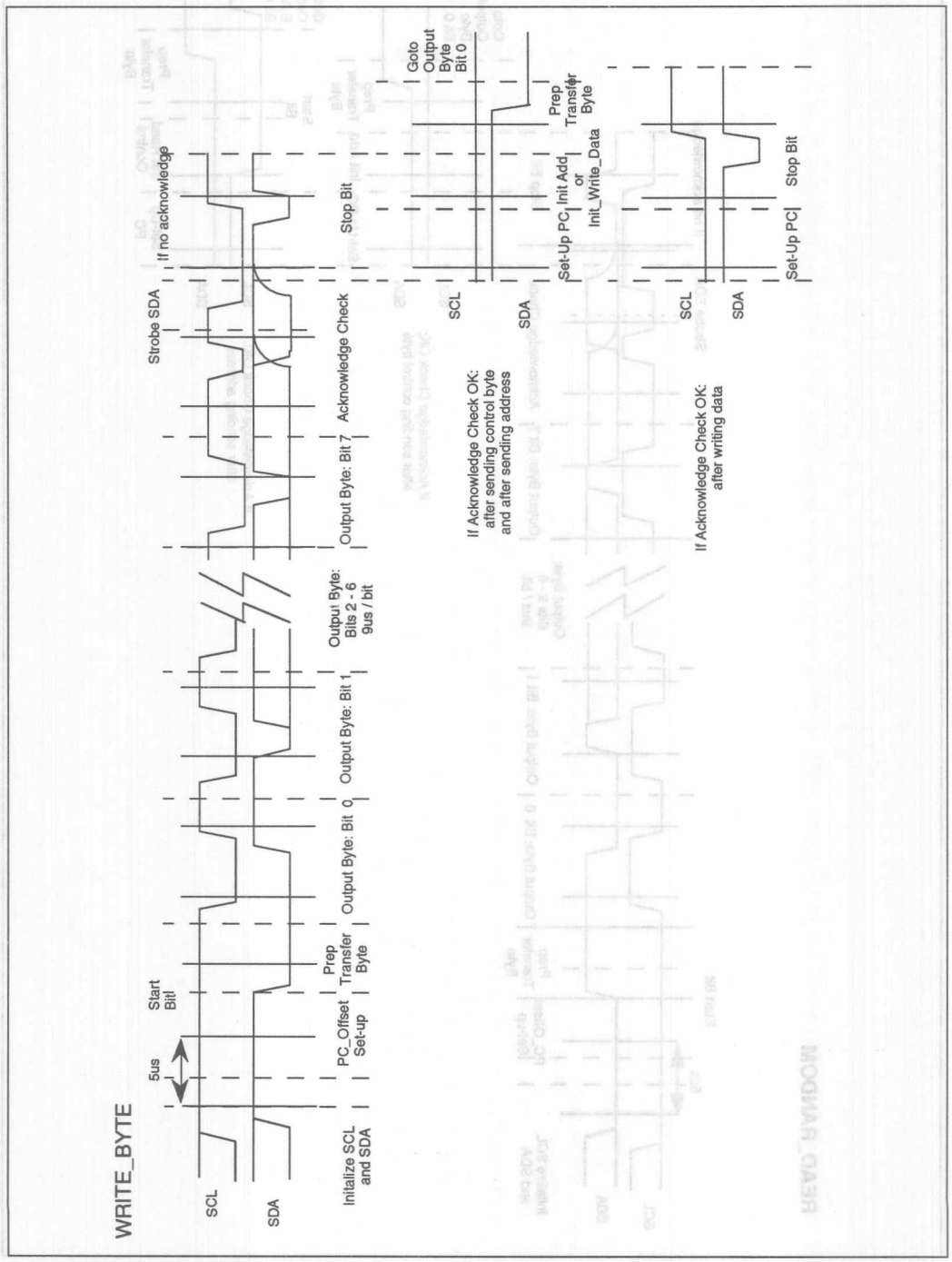
;************************  End of Program Loop  ************************
;*************************************************************************
INFINITELOOP
    MOVLW   0AAH    ; NOTE: Bit7 = '1' to let EEPROM finish writing, MTA85X1X
    MOVWF   PORTB
    CLRWDT
    GOTO    INFINITELOOP

;*************************************************************************
;*******************************  End MAIN  ******************************
   END
```


© 1994 Microchip Technology Inc.

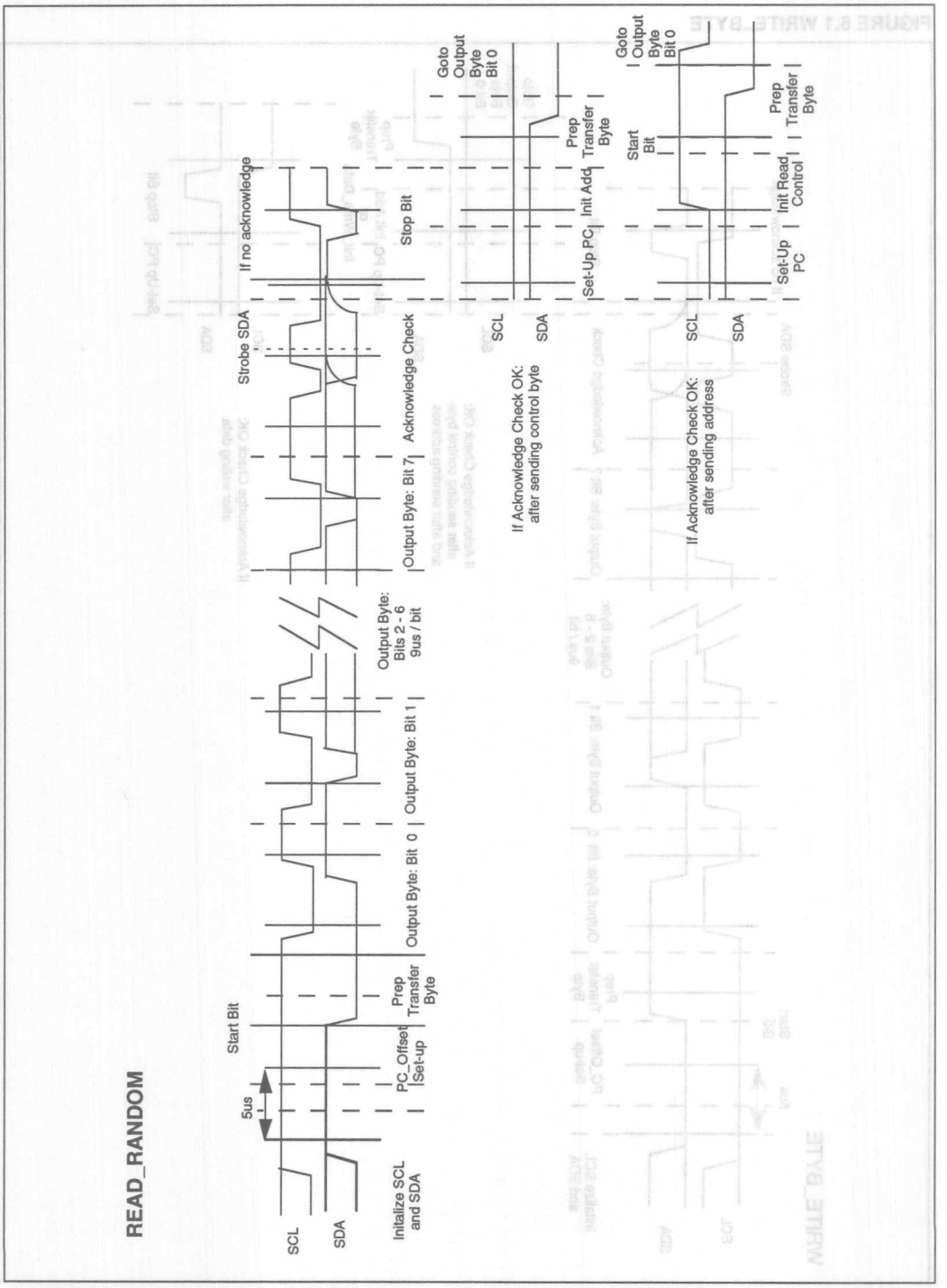
6.0 TIMING DIAGRAMS

FIGURE 6.1 WRITE_BYTE



6

FIGURE 6.2 READ_RANDOM



© 1994 Microchip Technology Inc.

FIGURE 6.2 READ_RANDOM (CONTINUED)

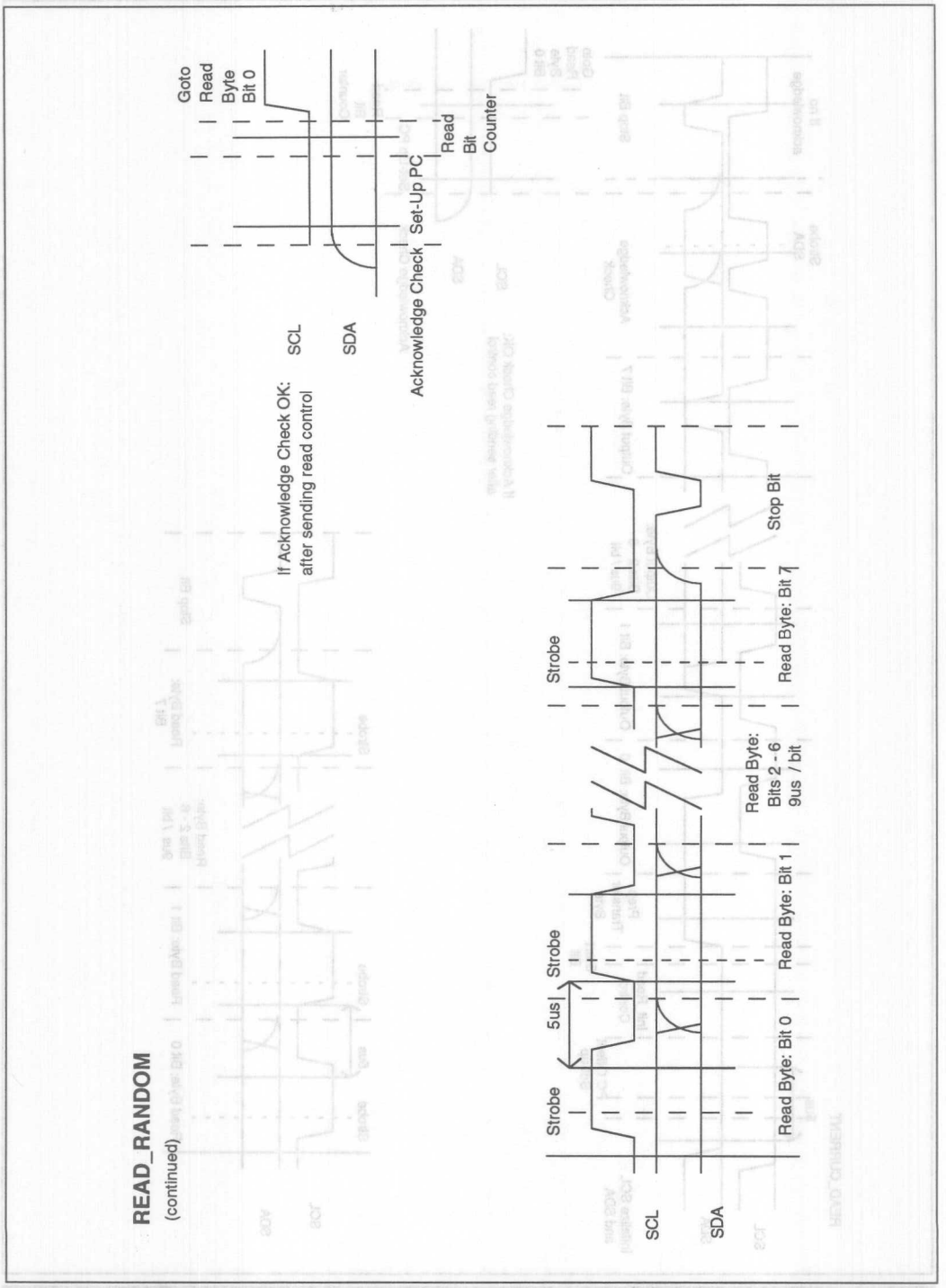
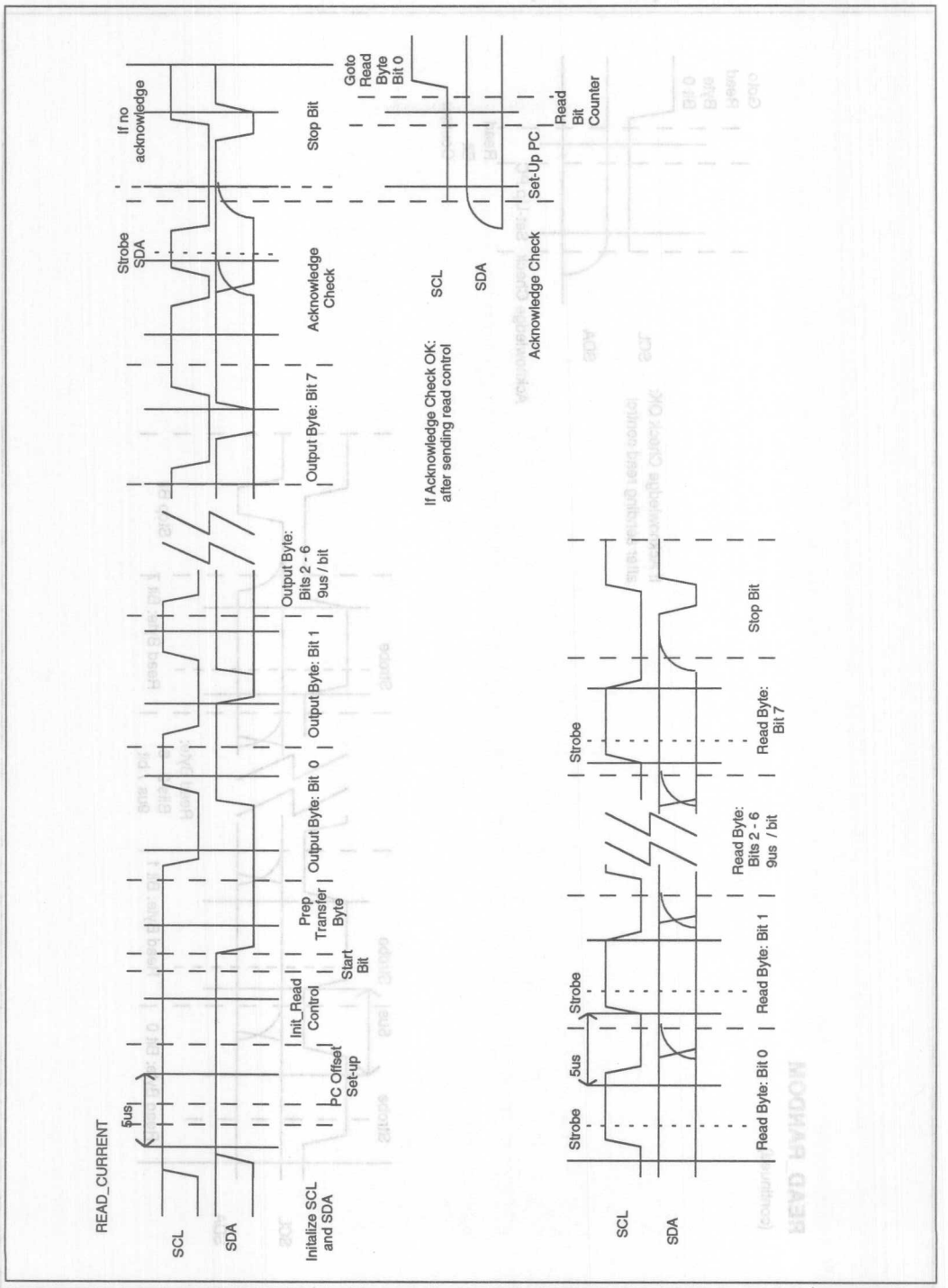


FIGURE 6.3 READ_CURRENT



© 1994 Microchip Technology Inc.

MICROCHIP

AN569

Hardware and Software Resolution For a Pointing Device

INTRODUCTION

The rated differences in pointing device resolution can be confusing to the user; this application note describes the method for calculating hardware resolution of a pointing device that incorporates Microchip's MTA41XXX Mouse Controller. It also includes an explanation of the software controlled resolution for these same devices. Mice and trackball resolution is rated in terms of DPI (Dots Per Inch). This resolution may be controlled via hardware or software, but specific differences exist in each case.

THEORY OF OPERATION

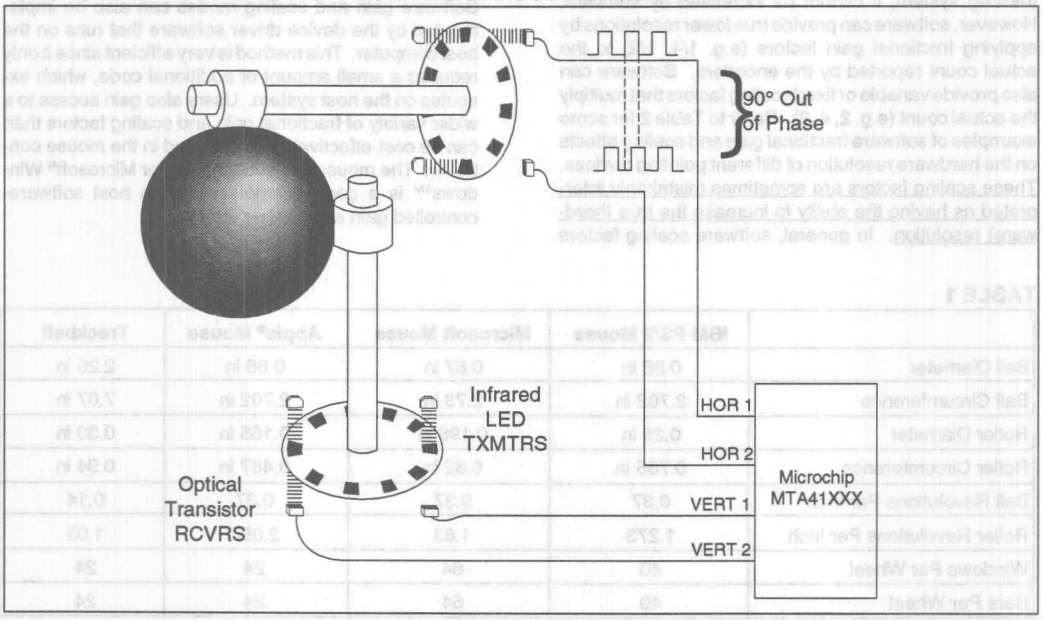
The basic hardware resolution or DPI of a mouse can be found using the following formula:

$$DPI = Rev\ per\ inch_{roller} * Logic\ states\ per\ rev$$

Refer to Figure 1 for a visual representation of the following material. A standard motion translator for mice is the use of two slotted wheels, one each for horizontal and vertical direction. Also, there are two optical receivers per slotted wheel. As the slotted wheel turns, infrared beams of light are alternately transmitted and blocked, thereby sending a series of ones and zeros to the optical transistor receivers. The two optical receivers are offset from each other such that the resulting signals are 90° out of phase. This phase difference results in two distinctly separate signals. The controller interprets what direction the mouse is moving along either axis by the order in which it receives these two signals. It should be noted here that the number of closed slots or bars on the wheel is equal to the number of open slots. From this information, the number of logic states per revolution is calculated as follows:

$$Logic\ states\ per\ rev = Optical\ RCVRS_{wheel} * (Windows_{wheel} + Bars_{wheel})$$

FIGURE 1 - TYPICAL ELECTRO-MECHANICAL MOUSE OPERATION



PS/2 is a registered trademark of IBM Corp.
Microsoft is a registered trademark of Microsoft Corp.
Windows is a trademark of Microsoft Corp.
Apple is a registered trademark of Apple Computer, Inc.

© 1993 Microchip Technology Inc.

Before we can calculate the number of roller revolutions per inch of mouse travel, the number of ball revolutions per inch of mouse travel must first be calculated. This is accomplished below by dividing the ball circumference, or pi times the ball diameter, into the unit of interest.

$$Revolution\ per\ inch_{ball} = 1\text{inch}/Circumference_{ball}$$

At this point, we can see that the number of roller revolutions per inch of mouse travel can be found by multiplying the number of ball revolutions per inch times the ratio derived from dividing ball circumference by roller circumference.

$$Revolution\ per\ inch_{roller} = Revolution\ per\ inch_{ball} * (Circumference_{ball}/Circumference_{roller})$$

Refer to Appendix A for an example of the above calculations using typical mouse specifications. Table 1 includes actual specifications and resulting DPI calculations using the information in this note.

SOFTWARE MODIFICATION OF DEVICE RESOLUTION

Now that the foundation for the basic physical (true) resolution of the mouse or trackball has been established, let's explore how this can be modified with software running on the host computer (device driver), or by the mouse controller. It should be noted here that most graphical user software is designed to operate most efficiently within the 200-400 DPI range.

Since the basic resolution of the device is set by the tracking system, it cannot be increased by software. However, software can provide true lower resolutions by applying fractional gain factors (e.g. 1/4, 1/2) to the actual count reported by the encoders. Software can also provide variable or fixed scaling factors that multiply the actual count (e.g. 2, 4, 8). Refer to Table 2 for some examples of software fractional gain and scaling effects on the hardware resolution of different pointing devices. These scaling factors are sometimes mistakenly interpreted as having the ability to increase the true (hardware) resolution. In general, software scaling factors that increased the number of counts reported per inch of mouse movement contribute to a loss of granularity, especially if a fixed scalar is applied. Variable scalars (often referred to as ballistic gain) that apply a multiplier based on the number of incoming counts can supply increased or decreased counts per inch without a degradation in granularity at low mouse velocities. The primary purpose of these scaling and resolution modes is to alter the mouse's motion sensitivity to suit the individual user.

If given a command from the host or some other means (e.g. hardware switch on the mouse), the mouse controller can also perform these tasks. The MTA41110 supports the command method, which is the most efficient in terms of hardware cost. When the MTA41300 is configured for the RS232 interface option, it is a transmit-only device, thereby not supporting the host-command mode. In the PS/2® interface mode the MTA41300 controller will respond to the PS/2 "*Set Resolution*" command. However, this response is only for software compatibility purposes, and the true resolution remains fixed.

The MTA41110 implements the complete IBM PS/2 specification, including the software fractional gain and variable scaling modes. The host driver software must be capable of issuing commands (e.g. "*Set Resolution*", "*Set Scaling*", etc.) to the MTA41110 controller in order for the user to benefit from these built-in firmware functions. The MTA41300 is a fixed-resolution device and any desired software resolution and scaling modes must be performed by the driver software.

Software gain and scaling modes can also be implemented by the device driver software that runs on the host computer. This method is very efficient since it only requires a small amount of additional code, which executes on the host system. Users also gain access to a wider variety of fractional gain and scaling factors than can be cost-effectively implemented in the mouse controller. The mouse control panel under Microsoft® Windows™ is a good example of such host software-controlled gain and scaling factors.

TABLE 1

	IBM PS/2 Mouse	Microsoft Mouse	Apple® Mouse	Trackball
Ball Diameter	0.86 in	0.87 in	0.86 in	2.25 in
Ball Circumference	2.702 in	2.73 in	2.702 in	7.07 in
Roller Diameter	0.25 in	0.196 in	0.155 in	0.30 in
Roller Circumference	0.785 in	0.62 in	0.487 in	0.94 in
Ball Revolutions Per Inch	0.37	0.37	0.37	0.14
Roller Revolutions Per Inch	1.273	1.63	2.054	1.05
Windows Per Wheel	40	64	24	24
Bars Per Wheel	40	64	24	24
Logic States Per Revolution	160	256	96	96
Dots Per Inch	203.718	417.07	197.147	101.09

© 1993 Microchip Technology Inc.

As a general note, the PS/2 specification may be somewhat confusing regarding device resolution. It specifies software-implemented resolution modes for a PS/2 system, including the encoder hardware. Also, it defines software commands, sent to the controller from the host, in terms of this absolute resolution. For example, the options for the "*Set Resolution*" commands are 1, 2, 4 and 8 counts per millimeter (25, 50, 100 and 200 DPI). These options actually instruct the mouse controller to divide the incoming count by 8, 4, 2, or 1 respectively, then report the divided count to the host. The actual hardware resolution of the IBM PS/2 mouse is 200 DPI. Therefore, for mice whose hardware resolutions are not 200 DPI, the "*Set Resolution*" command will not be the actual available resolution, rather the available resolutions will be ratioed relative to a 200 DPI (hardware) mouse.

DYNAMIC RESPONSE

There is a maximum linear velocity at which the mouse can be moved and be 100 percent effective. Operating the mouse faster than this will result in missed quadrature information. This section will show the derivation of that maximum velocity.

According to the MTA41110 data sheet, the HOR and VERT outputs are sampled at ~8700 samples per second.

Sample Rate = Frequency
Period = 1/Frequency = 1/8700 Hz = 114.94µsec

The hardware interface of the Microsoft mouse which was referred to earlier will be used as the example for this discussion. It has 64 slots per wheel, which results in 256 logic states per wheel revolution. The equation below displays the method for calculating the wheel's maximum rotation rate.

$$Period_{wheel} = (256 * 114.94\ \mu sec) = 29.42\ msec$$

$$Frequency_{wheel} = 1/Period_{wheel} = 33.98\ rev/sec$$

Since the roller and wheel are attached on the same shaft in a one-to-one relationship, the frequency of the roller must also equal 33.98 revolutions per second.

The velocity of the roller can then be found with the following equation. Remember, the roller radius is .098 inches. Refer to the Microsoft mouse section in Table 1 for this information.

$$Velocity_{roller} = 2 * Radius_{roller} * Frequency_{roller} = 20.9\ in/sec$$

Because there is a direct correlation between mouse travel and travel along the surface of the tracking ball, the velocity of the mouse can also not exceed 20.9 inches per second.

CONCLUSION

From the calculations in the body of this note, it is shown that the mouse controller has no effect on the resolution of the mouse or trackball. DPI is completely a function of mechanical design.

APPENDIX A

These detailed calculations show how the resolution of a typical mouse is determined. They show how the physical design of the motion tracking system (encoder wheels and tracking rollers) determine the basic (hardware) resolution of the mouse. This analysis assumes that the mouse controller can report one count to the system for each logic transition at the motion encoders. This applies directly to the MTA41300 and MTA41110 Mouse and Trackball controllers since they both contain modes that report one count for each motion encoder state. The MTA41110 also contains software resolution modes which will be discussed later in this analysis.

First, let's begin by sizing a typical tracking ball and roller. The roller is a small diameter wheel placed in contact with the main ball.

$$Diameter_{ball} = 0.86\ in$$

$$Circumference_{ball} = \pi * Diameter_{ball} = 2.702\ in$$

$$Diameter_{roller} = 0.159\ in$$

$$Circumference_{roller} = \pi * Diameter_{roller} = 0.5\ in$$

Now let's see how many revolutions the tracking roller makes in one inch of mouse movement.

$$Revolution\ per\ inch_{ball} = 1 inch / Circumference_{ball} = 0.37$$

$$Revolution\ per\ inch_{roller} = Revolution\ per\ inch_{ball} * (Circumference_{ball} / Circumference_{roller}) = 2.002$$

TABLE 2

	True DPI	x2	x4	x0.5	x0.25
IBM PS/2 Mouse	200	400	800	100	50
Microsoft Mouse	400	800	1600	200	100
Apple Mouse	200	400	800	100	50
Trackball	100	200	400	50	25

NOTES: The examples given in this application note are:
1. Hardware examples for illustration purposes only.
2. Do not imply the use of MTA41XXX devices by the respective manufacturers.

© 1993 Microchip Technology Inc.

Notice that the size of the tracking ball has no effect on the number of revolutions the tracking roller makes in one inch. This is because one inch of mouse movement corresponds directly to one inch of movement along the circumference of the ball. However, roller size does affect basic resolution.

Let's examine the physical design of some normal and high resolution encoders, and see why they are a key component in determining basic resolution. The roller and an encoder wheel are usually mounted axially on a single shaft, resulting in a one-to-one relationship between the roller and encoder wheel.

Optical receivers per wheel = 2

To determine the direction of movement, quadrature output is needed from the encoders. Standard industry practice is to use two optical detectors (e.g. phototransistors) for this purpose.

Windows per wheel = 25
Bars per wheel = 25

The number of windows and bars in the encoder wheels determines the number of pulses per revolution from the encoder.

Now let's calculate the basic physical revolution in DPI (Dots Per Inch)

Logic states per rev = $Optical\ Rcvrs_{wheel}$ * ($Windows_{wheel}$+$Bars_{wheel}$) = 100

The basic resolution in DPI can now be calculated as:

DPI = $Rev\ per\ inch_{roller}$ * Logic states per rev = 200.195

Now let's analyze a typical high resolution mouse. Typically, "high" resolution mice and trackballs contain more windows in the encoder wheels. This results in higher resolution since more pulses are created for each revolution of the wheel.

Windows per wheel = 50
Bars per wheel = 50

Logic states per rev = $Optical\ Rcvrs_{wheel}$ * ($Windows_{wheel}$+$Bars_{wheel}$) = 200

DPI = $Rev\ per\ inch_{roller}$ * Logic states per rev = 400.39

Doubling the number of windows in the encoder wheel(s) converts a true 200DPI mouse into a true 400 DPI mouse.

© 1993 Microchip Technology Inc.

SECTION 7
INTERFACING PIC16/17 WITH SERIAL EEPROMS

Communicating with I²C Bus Using the PIC16C5X - AN515 ..... 7- 1
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X Microcontrollers- AN530 ..... 7- 11
Logic Powered Serial EEPROMs - AN535 ..... 7- 29

7

© 1994 Microchip Technology Inc.

MICROCHIP

AN515

Communicating with the I²C™ Bus Using the PIC16C5X

INTRODUCTION

The Microchip Technology Inc.'s 24CXX and 85CXX Serial EEPROMs feature a two wire serial interface bus. The bus protocol is I²C compatible. Interface to a serial port with I²C bus protocol in a microcontroller is trivial. This application note is intended for design engineers who want to develop their software programs to communicate a microcontroller with a 2-wire bus Serial EEPROM through a general purpose I/O port.

Unlike the 3-wire bus Serial EEPROMs, the 24CXX/85CXX communicate with any microcontroller only by a serial data I/O line (SDA) and a serial clock (SCL). Chip select is not required. Data transfer may be initiated only when the bus is not busy. During such transfer, the data line (SDA) must remain stable whenever the clock line (SCL) is high. Changes in the data line while the clock line is high are interpreted as a START or STOP condition. A typical transfer format is shown in Figure 1.

After the START condition, a slave address is sent. This address is 7-bits long, the eighth bit is a data direction bit. (R/$\overline{W}$ - a logical '0' indicates a transmission WRITE, a logical '1' represents a request for data READ. A data transfer is always terminated by a STOP condition generated by the master controller. However, if a master still wishes to communicate on the bus, it can generate another START condition and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such transfer.

FIGURE 1 - TRANSFER FORMAT

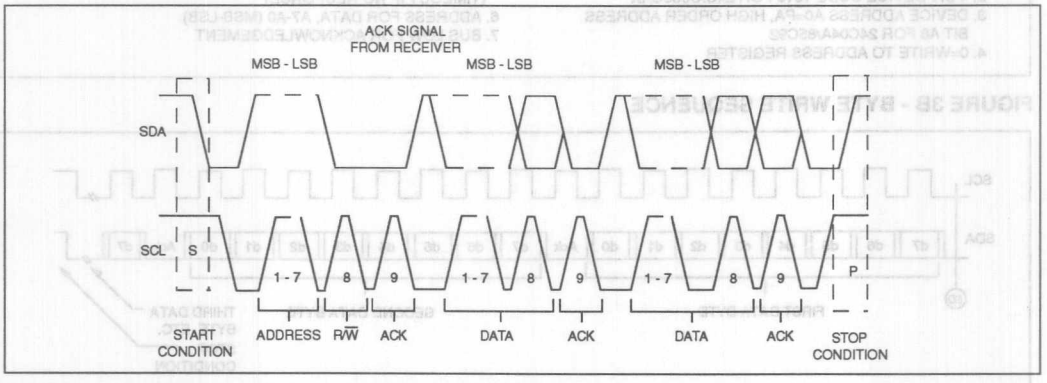
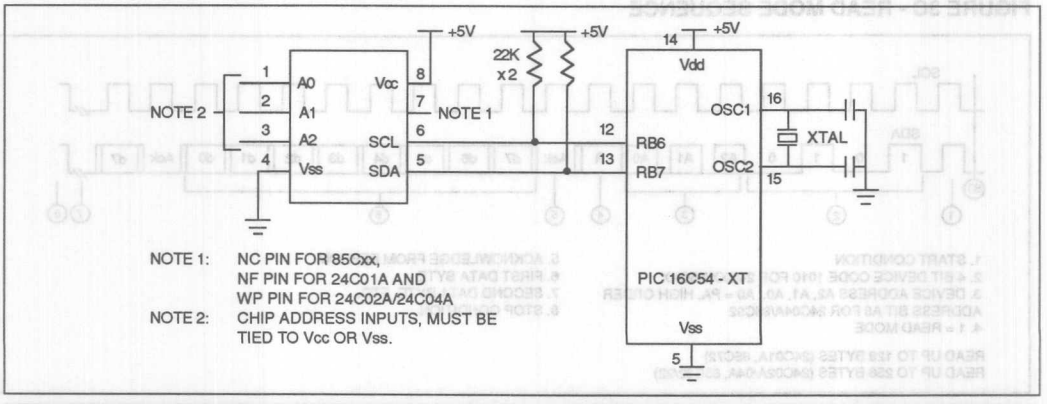


FIGURE 2 - A SIMPLE HARDWARE CONNECTION



7

© 1993 Microchip Technology Inc.

Communicating with the I²C™ Bus Using the PIC16C5X

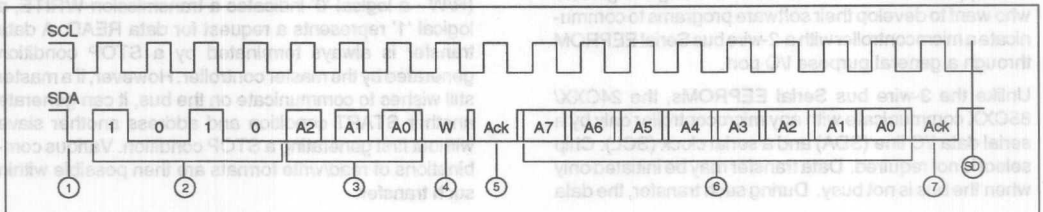
An example program has been provided in Appendix A containing all PIC16C54 routines needed to exercise a 24CXX or 85CXX device. A simple hardware connection is illustrated in Figure 2. A maximum of eight 24C01A/24C02A/85C72/85C82's, or four 24C04A/85C92's can be addressed by a microcontroller on the same two wire bus without additional interfaces. Each device is identified by its Chip Address and will only respond to the correct slave address. A detailed bus flow is shown in Figure 3.

Figure 3 as shown below describes how the bit stream is set up for READ and WRITE mode in the microcomputer programming software prior to sending it on the two wire serial bus.

The stop condition, after the write sequence, starts the internal self-timed write cycle which may last up to 6 milliseconds (.7 ms per byte). Acknowledge signal should be monitored during this period.

Author: *Bruce Negley*
Memory Products Division

FIGURE 3A - SETTING THE INTERNAL WORD ADDRESS OF THE 24CXX/85CXX



1. START CONDITION
2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX
3. DEVICE ADDRESS A0=PA, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92
4. 0=WRITE TO ADDRESS REGISTER
5. EEPROM DRIVES BUS LOW WITH ACKNOWLEDGEMENT (TIMEOUT IF NO RESPONSE)
6. ADDRESS FOR DATA, A7-A0 (MSB-LSB)
7. BUS LOW FOR ACKNOWLEDGEMENT

FIGURE 3B - BYTE WRITE SEQUENCE

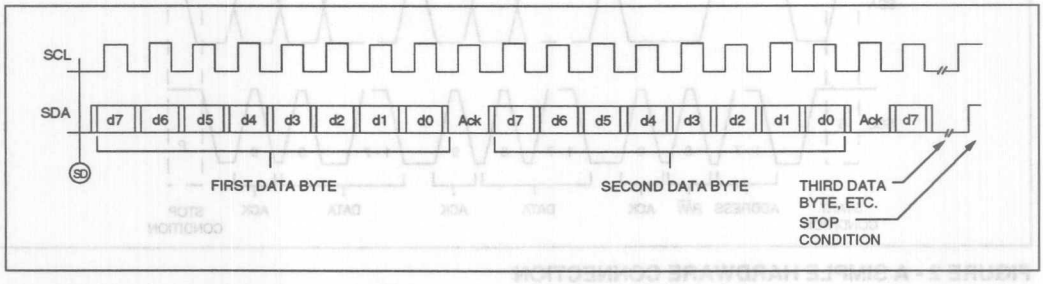
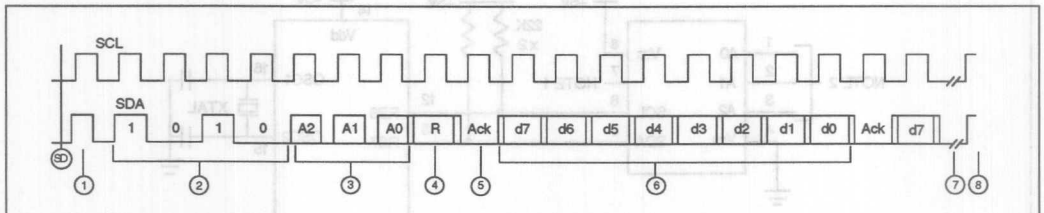


FIGURE 3C - READ MODE SEQUENCE



1. START CONDITION
2. 4 BIT DEVICE CODE 1010 FOR 24CXX/85CXX
3. DEVICE ADDRESS A2, A1, A0. A0 = PA, HIGH ORDER ADDRESS BIT A8 FOR 24C04A/85C92
4. 1 = READ MODE

READ UP TO 128 BYTES (24C01A, 85C72)
READ UP TO 256 BYTES (24C02A/04A, 85C82/92)

5. ACKNOWLEDGE FROM EEPROM
6. FIRST DATA BYTE
7. SECOND DATA BYTE, ETC.
8. STOP CONDITION

© 1993 Microchip Technology Inc.

Appendix A:

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0              PAGE  1
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         TITLE " TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x "
                     0002 ;
                     0003         LIST P=16C54
                     0004 ;
                     0005
;*****************************************************************************
                     0006 ;**  Two wire/I2C Bus READ/WRITE Sample Routines of Microchip's
                     0007 ;**  24Cxx / 85Cxx serial CMOS EEPROM interfacing to a
                     0008 ;**  PIC16C54 8-bit CMOS single chip microcomputer
                     0009 ;**  Revised Version 2.1 (8/16/94).
                     0010 ;**
                     0011 ;**      Part use = PIC16C54-XT/JW
                     0012 ;**  Note: 1) All timings are based on a reference crystal frequency of 2MHz
                     0013 ;**            which is equivalent to an instruction cycle time of 2 usec.
                     0014 ;**        2) Address and literal values are read in octal unless otherwise
                     0015 ;**           specified.
                     0016
;*****************************************************************************
                     0017 ;
                     0018 ;-----------------------------------------------
                     0019 ;       Files Assignment
                     0020 ;-----------------------------------------------
                     0021 ;
0002                 0022 PC      EQU     2              ; Program counter
0003                 0023 STAT    EQU     3              ; PIC status byte
0004                 0024 FSR     EQU     4              ; File Select Register
0005                 0025 RA      EQU     5              ; Port A use to select device address
0006                 0026 RB      EQU     6              ; RB7 = SDA, RB6 = SCL
                     0027 ;
0008                 0028 STATUS  EQU     08             ; Status register
0009                 0029 FLAG    EQU     09             ; Common flag bits register
000A                 0030 EEPROM  EQU     0A             ; Bit buffer
000B                 0031 ERCODE  EQU     0B             ; Error code (to indicate bus status)
0010                 0032 ADDR    EQU     10             ; Address register
0011                 0033 DATAI   EQU     11             ; Stored data input register
0012                 0034 DATAO   EQU     12             ; Stored data output register
0013                 0035 SLAVE   EQU     13             ; Device address (1010xxx0)
0014                 0036 TXBUF   EQU     14             ; TX buffer
0015                 0037 RXBUF   EQU     15             ; RX buffer
0016                 0038 COUNT   EQU     16             ; Bit counter
                     0039 ;
0018                 0040 TIMER0  EQU     18             ; Delay timer0
0019                 0041 TIMER1  EQU     19             ; Delay timer1
                     0042 ;
                     0043 ;
                     0044 ;-----------------------------------------------
                     0045 ;                Bit Assignments
                     0046 ;-----------------------------------------------
                     0047 ;
                     0048 ; Status bits
                     0049 ;
0002                 0050 Z       EQU     2
0000                 0051 C       EQU     0
                     0052 ;
                     0053 ; FLAG Bits
```

7

© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0                PAGE  2
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0054 ;
0000                 0055 ERRORF  EQU     0               ; Error flag
                     0056 ;
                     0057 ; EEPROM Bits
                     0058 ;
0007                 0059 DI      EQU     7               ; EEPROM input
0006                 0060 DO      EQU     6               ; EEPROM output
                     0061 ;
                     0062 ; I2C Device Bits
                     0063 ;
0007                 0064 SDA     EQU     7               ; RB7, data in/out
0006                 0065 SCL     EQU     6               ; RB6, serial clock
                     0066 ;
                     0067 ;END FILES/BITS EQUATE
                     0068 ;
                     0069 ;-------------------------------------------------
                     0070 ;      Two wire/I2C - CPU communication error status table and subroutine
                     0071 ;-------------------------------------------------
                     0072 ;  input  :     W-reg   = error code
                     0073 ;  output :     ERCODE  = error code
                     0074 ;           FLAG(ERROR) = 1
                     0075 ;
                     0076 ;         code          error status mode
                     0077 ;        ------  ------------------------------------
                     0078 ;           1    :  SCL locked low by device (bus is still busy)
                     0079 ;           2    :  SDA locked low by device (bus is still busy)
                     0080 ;           3    :  No acknowledge from device (no handshake)
                     0081 ;           4    :  SDA bus not released for master to generate STOP bit
                     0082 ;-------------------------------------------------
                     0083 ;
                     0084 ;Subroutine to identify the status of the serial clock (SCL) and serial data
                     0085 ;(SDA) condition according to the error status table. Codes generated are
                     0086 ;useful for bus/device diagnosis.
                     0087 ;
                     0088 ERR
0000 0709            0089         BTFSS   FLAG,ERRORF     ; Remain as first error encountered
0001 002B            0090         MOVWF   ERCODE          ; Save error code
0002 0509            0091         BSF     FLAG,ERRORF     ; Set error flag
0003 0800            0092         RETLW   0
                     0093 ;
                     0094 ;-------------------------------------------------
                     0095 ;        START bus communication routine
                     0096 ;-------------------------------------------------
                     0097 ;        input   : none
                     0098 ;        output  : initialize bus communication
                     0099 ;-------------------------------------------------
                     0100 ;
                     0101 ;Generate START bit (SCL is high while SDA goes from high to low transition)
                     0102 ;and check status of the serial clock.
                     0103 BSTART
0004 0C3F            0104         MOVLW   B'00111111'     ; Put SCL, SDA line in output state
0005 0006            0105         TRIS    RB
0006 05E6            0106         bsf     RB,SDA          ;make sure sda is high
```


© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0          PAGE  3
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

0007 05C6            0107          BSF     RB,SCL          ; Set clock high
0008 0C01            0108          MOVLW   1               ; Ready error status code 1
0009 07C6            0109          BTFSS   RB,SCL          ; Locked?
000A 0900            0110          CALL    ERR             ; SCL locked low by device
000B 04E6            0111          BCF     RB,SDA          ; SDA goes low during SCL high
000C 0000            0112          NOP                     ; Timing adjustment
000D 0000            0113          NOP
000E 0000            0114          NOP
000F 04C6            0115          BCF     RB,SCL          ; Start clock train
0010 0800            0116          RETLW   0
                     0117 ;
                     0118 ;END SUB
                     0119 ;
                     0120 ;-------------------------------------------
                     0121 ;       STOP bus communication routine
                     0122 ;-------------------------------------------
                     0123 ;       Input   :       None
                     0124 ;       Output  :       Bus communication, STOP condition
                     0125 ;-------------------------------------------
                     0126 ;
                     0127 ;Generate STOP bit (SDA goes from low to high during SCL high state)
                     0128 ;and check bus conditions.
                     0129 ;
                     0130 BSTOP
0011 0C3F            0131          MOVLW   B'00111111'     ; Put SCL, SDA line in output state
0012 0006            0132          TRIS    RB
0013 04E6            0133          BCF     RB,SDA          ; Return SDA to low
0014 05C6            0134          BSF     RB,SCL          ; Set SCL high
0015 0000            0135          nop
0016 0000            0136          nop
0017 0000            0137          nop
0018 0C01            0138          MOVLW   1               ; Ready error code 1
0019 07C6            0139          BTFSS   RB,SCL          ; High?
001A 0900            0140          CALL    ERR             ; No, SCL locked low by device
001B 05E6            0141          BSF     RB,SDA          ; SDA goes from low to high during SCL high
001C 0C04            0142          MOVLW   4               ; Ready error code 4
001D 07E6            0143          BTFSS   RB,SDA          ; High?
001E 0900            0144          CALL    ERR             ; No, SDA bus not release for STOP
001F 0800            0145          RETLW   0
                     0146 ;
                     0147 ;END SUB
                     0148 ;
                     0149 ;-------------------------------------------
                     0150 ;       Serial data send from PIC to serial EEPROM, bit-by-bit subroutine
                     0151 ;-------------------------------------------
                     0152 ;       Input   :       None
                     0153 ;       Output  :       To (DI) of serial EEPROM device
                     0154 ;-------------------------------------------
                     0155 ;
                     0156 BITIN
0020 0CBF            0157          MOVLW   B'10111111'     ; Force SDA line as input
0021 0006            0158          TRIS    RB
0022 05E6            0159          BSF     RB,SDA          ; Set SDA for input
```

7

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0          PAGE  4
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

0023 04EA            0160          BCF     EEPROM,DI
0024 05C6            0161          BSF     RB,SCL          ; Clock high
0025 0C01            0162          MOVLW   1
0026 06C6            0163          BTFSC   RB,SCL          ; Skip if SCL  is high
0027 0A2B            0164          GOTO    BIT1
0028 0709            0165          BTFSS   FLAG,ERRORF     ; Remain as first error encountered
0029 002B            0166          MOVWF   ERCODE          ; Save error code
002A 0509            0167          BSF     FLAG,ERRORF     ; Set error flag
                     0168 BIT1
002B 06E6            0169          BTFSC   RB,SDA          ; Read SDA pin
002C 05EA            0170          BSF     EEPROM,DI       ; DI = 1
002D 0000            0171          NOP                     ; Delay
002E 04C6            0172          BCF     RB,SCL          ; Return SCL to low
002F 0800            0173          RETLW   0
                     0174 ;
                     0175 ;END SUB
                     0176 ;
                     0177 ;-----------------------------------------------------
                     0178 ;       Serial data receive from serial EEPROM to PIC, bit-by-bit subroutine
                     0179 ;-----------------------------------------------------
                     0180 ;       Input   :       EEPROM file
                     0181 ;       Output  :       From (DO) of serial EEPROM device to PIC
                     0182 ;-----------------------------------------------------
                     0183 ;
                     0184 BITOUT
0030 0C3F            0185          MOVLW   B'00111111'     ; Set SDA, SCL as outputs
0031 0006            0186          TRIS    RB
0032 07CA            0187          BTFSS   EEPROM,DO
0033 0A3C            0188          GOTO    BIT0
0034 05E6            0189          BSF     RB,SDA          ; Output bit 0
0035 0C02            0190          MOVLW   2
0036 06E6            0191          BTFSC   RB,SDA          ; Check for error code 2
0037 0A40            0192          GOTO    CLK1
0038 0709            0193          BTFSS   FLAG,ERRORF     ; Remain as first error encountered
0039 002B            0194          MOVWF   ERCODE          ; Save error code
003A 0509            0195          BSF     FLAG,ERRORF     ; Set error flag
003B 0A40            0196          GOTO    CLK1            ; SDA locked low by device
                     0197 ;
                     0198 BIT0
003C 04E6            0199          BCF     RB,SDA          ; Output bit 0
003D 0000            0200          NOP                     ; Delay
003E 0000            0201          NOP
003F 0000            0202          NOP
                     0203 CLK1
0040 05C6            0204          BSF     RB,SCL
0041 0C01            0205          MOVLW   1               ; Error code 1
0042 06C6            0206          BTFSC   RB,SCL          ; SCL locked low?
0043 0A47            0207          GOTO    BIT2            ; No.
0044 0709            0208          BTFSS   FLAG,ERRORF     ; Yes.
0045 002B            0209          MOVWF   ERCODE          ; Save error code
0046 0509            0210          BSF     FLAG,ERRORF     ; Set error flag
                     0211 BIT2
0047 0000            0212          NOP
```


© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0          PAGE  5
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

0048 0000            0213          NOP
0049 04C6            0214          BCF     RB,SCL          ; Return SCL to low
004A 0800            0215          RETLW   0
                     0216 ;
                     0217 ;END SUB
                     0218 ;
                     0219 ;
                     0220 ;-------------------------------------------
                     0221 ;        RECEIVE          DATA     subroutine
                     0222 ;-------------------------------------------
                     0223 ;        Input    :       None
                     0224 ;        Output   :       RXBUF = Receive 8-bit data
                     0225 ;-------------------------------------------
                     0226 ;
                     0227 RX
004B 0C08            0228          MOVLW   .8              ; 8 bits of data
004C 0036            0229          MOVWF   COUNT
004D 0075            0230          CLRF    RXBUF
                     0231 ;
                     0232 RXLP
004E 0375            0233          RLF     RXBUF           ; Shift data to buffer
004F 0703            0234          SKPC
0050 0415            0235          BCF     RXBUF,0         ; carry -> f(0)
0051 0603            0236          SKPNC
0052 0515            0237          BSF     RXBUF,0
0053 0920            0238          CALL    BITIN
0054 06EA            0239          BTFSC   EEPROM,DI
0055 0515            0240          BSF     RXBUF,0         ; Input bit =1
0056 02F6            0241          DECFSZ  COUNT           ; 8 bits?
0057 0A4E            0242          GOTO    RXLP
0058 05CA            0243          BSF     EEPROM,DO       ; Set acknowledge bit = 1
0059 0930            0244          CALL    BITOUT          ; to STOP further input
005A 0800            0245          RETLW   0
                     0246 ;
                     0247 ;END SUB
                     0248 ;
                     0249 ;-------------------------------------------
                     0250 ;        TRANSMIT         DATA     subroutine
                     0251 ;-------------------------------------------
                     0252 ;        Input    :       TXBUF
                     0253 ;        Output   :       Data X'mitted to EEPROM device
                     0254 ;-------------------------------------------
                     0255 ;
                     0256 TX
005B 0C08            0257          MOVLW   .8
005C 0036            0258          MOVWF   COUNT
                     0259 ;
                     0260 TXLP
005D 04CA            0261          BCF     EEPROM,DO       ; Shift data bit out.
005E 06F4            0262          BTFSC   TXBUF,7         ; If shifted bit = 0, data bit = 0
005F 05CA            0263          BSF     EEPROM,DO       ; Otherwise data bit = 1
0060 0930            0264          CALL    BITOUT          ; Serial data out
0061 0374            0265          RLF     TXBUF           ; Rotate TXBUF left
```

7

© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0                  PAGE  6
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

0062 0703            0266          SKPC                    ; f(6) -> f(7)
0063 0414            0267          BCF     TXBUF,0         ; f(7) -> carry
0064 0603            0268          SKPNC                   ; carry -> f(0)
0065 0514            0269          BSF     TXBUF,0
0066 02F6            0270          DECFSZ  COUNT           ; 8 bits done?
0067 0A5D            0271          GOTO    TXLP            ; No.
0068 0920            0272          CALL    BITIN           ; Read acknowledge bit
0069 0C03            0273          MOVLW   3
006A 06EA            0274          BTFSC   EEPROM,DI       ; Check for acknowledgement
006B 0900            0275          CALL    ERR             ; No acknowledge from device
006C 0800            0276          RETLW   0
                     0277 ;
                     0278 ;END SUB
                     0279 ;
                     0280 ;-------------------------------------------
                     0281 ;       BYTE-WRITE, write one byte to EEPROM device
                     0282 ;-------------------------------------------
                     0283 ;       Input   :       DATAO= data to be written
                     0284 ;                       ADDR    = destination address
                     0285 ;                       SLAVE   = device address (1010xxx0)
                     0286 ;       Output  :       Data written to EEPROM device
                     0287 ;-------------------------------------------
                     0288 ;
                     0289          ORG     080             ; The location for BYTE-WRITE routine can be
                     0290 ;                                 ; assigned anywhere between (377-777) octal.
                     0291 WRBYTE
0080 0213            0292          MOVF    SLAVE,W         ; Get SLAVE address
0081 0034            0293          MOVWF   TXBUF           ; to TX buffer
0082 0904            0294          CALL    BSTART          ; Generate START bit
0083 095B            0295          CALL    TX              ; Output SLAVE address
0084 0210            0296          MOVF    ADDR,W          ; Get WORD address
0085 0034            0297          MOVWF   TXBUF           ; into buffer
0086 095B            0298          CALL    TX              ; Output WORD address
0087 0212            0299          MOVF    DATAO,W         ; Move DATA
0088 0034            0300          MOVWF   TXBUF           ; into buffer
0089 095B            0301          CALL    TX              ; Output DATA and detect acknowledgement
008A 0911            0302          CALL    BSTOP           ; Generate STOP bit
008B 0AD8            0303          goto    wrt_end
                     0304 ;
                     0305 ;
                     0306 ;
                     0307 ;-------------------------------------------
                     0308 ;       BYTE-READ, read one byte from serial EEPROM device
                     0309 ;-------------------------------------------
                     0310 ;       Input   :       ADDR    = source address
                     0311 ;                       SLAVE   = device address (1010xxx0)
                     0312 ;       Output  :       DATAI   = data read from serial EEPROM
                     0313 ;-------------------------------------------
                     0314 ;
                     0315          ORG     0C0             ; The location for BYTE-READ routine can be
                     0316 ;                                 ; assigned anywhere between (377-777) octal.
                     0317 RDBYTE
00C0 0213            0318          MOVF    SLAVE,W         ; Move SLAVE address
```


© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int   I2CBUS.ASM   8-17-1994  15:17:0  PAGE  7
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

00C1 0034            0319         MOVWF   TXBUF           ; into buffer (R/W = 0)
00C2 0904            0320         CALL    BSTART          ; Generate START bit
00C3 095B            0321         CALL    TX              ; Output SLAVE address. Check ACK.
00C4 0210            0322         MOVF    ADDR,W          ; Get WORD address
00C5 0034            0323         MOVWF   TXBUF
00C6 095B            0324         CALL    TX              ; Output WORD address. Check ACK.
00C7 0904            0325         CALL    BSTART          ; START READ (if only one device is
00C8 0213            0326         MOVF    SLAVE,W         ; connected to the I2C bus)
00C9 0034            0327         MOVWF   TXBUF
00CA 0514            0328         BSF     TXBUF,0         ; Specify READ mode (R/W = 1)
00CB 095B            0329         CALL    TX              ; Output SLAVE address
00CC 094B            0330         CALL    RX              ; READ in data and acknowledge
00CD 0911            0331         CALL    BSTOP           ; Generate STOP bit
00CE 0215            0332         MOVF    RXBUF,W         ; Save data from buffer
00CF 0031            0333         MOVWF   DATAI           ; to DATAI file.
00D0 0ADA            0334         goto    rd_end
                     0335 ;
                     0336 ;Test program to read and write ramdom
                     0337 start
00D1 0CAE            0338         movlw   0AE             ;set A2=A1=A0=1
00D2 0033            0339         movwf   SLAVE           ;       /
00D3 0C02            0340         movlw   2               ;set r/w loc. = 2
00D4 0030            0341         movwf   ADDR            ;       /
00D5 0C55            0342         movlw   55              ;write 55 to SEEPROM
00D6 0032            0343         movwf   DATAO           ;       /
00D7 0A80            0344         goto    WRBYTE          ;write a byte
                     0345 wrt_end
00D8 09DF            0346         call    delay           ;wait for write
                     0347                                 ;operation (internal)
00D9 0AC0            0348         goto    RDBYTE          ;read back data
                     0349 rd_end
00DA 0C55            0350         movlw   55              ;test if read
00DB 0191            0351         xorwf   DATAI,W         ;correct?
00DC 0743            0352         btfss   STAT,Z          ;yes then skip
                     0353 wrong
00DD 0ADD            0354         goto    wrong
                     0355 correct
00DE 0ADE            0356         goto    correct
                     0357 ;
                     0358 ;At 2.0Mhz, delay = approx. 3mS.
                     0359 delay
00DF 007F            0360         clrf    1F              ;clear last location
                     0361 dly1
00E0 0000            0362         nop
00E1 0000            0363         nop
00E2 0000            0364         nop
00E3 02FF            0365         decfsz  1F              ;reduce count
00E4 0AE0            0366         goto    dly1            ;loop
00E5 0800            0367         retlw   0
                     0368 ;
                     0369         org     0x1FF
01FF 0AD1            0370         goto    start
                     0371 ;
```


© 1993 Microchip Technology Inc.

```
MPASM 01.01.05 Int    I2CBUS.ASM   8-17-1994  15:17:0          PAGE  8
 TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x

LOC  OBJECT CODE     LINE SOURCE TEXT

                       0372          END
                       0373
```


© 1993 Microchip Technology Inc.

MICROCHIP

AN530

Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers

INTRODUCTION

Microchip Technology Inc.'s popular 93C46/56/66 and 93LC46/56/66 Serial EEPROMs feature a three/four wire serial interface bus. The attractive price and simple interface make it the ideal device for additional memory space. This application note is intended for design engineers who wish to incorporate a pre-packaged serial EEPROM interface driver into their application.

THE HARDWARE CONNECTION

A typical 4-wire hardware connection is illustrated in Figure 1A and a typical 3-wire connection is illustrated in Figure 1B. Since all I/O ports on the PIC16C5X are configurable as input and/or output, a 3-wire interface makes optimum utilization of the I/O pins by having a common connection for the DI and DO lines of the Serial EEPROM. The port pin on the PIC16C5X connected to these pins, has a default setting as an output and is configured, when needed, as an input during program execution.

THE SOFTWARE CONNECTION

An example interface driver is listed in Appendix A. A flow diagram is given in Figure 2. The interface driver is written to minimize both overhead to the calling program as well as the program space necessary for its inclusion into the user's code. The driver has been written as a generic driver to service all 93 Series Serial EEPROMs made by Microchip. Processor resources which must be made available to the driver prior to being called are: 1) Two levels of processor stack. 2) Six register locations (four for command/data passing and two for software counters). 3) The File Select Register (FSR), which is used to pass a command/data string pointer to the driver.

Note: The four command/data passing registers have to be defined consecutively in order for the FSR to access them successfully in the program execution.

The user should take the following steps when using the routines provided in Appendix A.

A) Specify and define a 3-/4-wire interface by defining the common connection to the DI/DO lines and setting the equate 'wire3' TRUE or FALSE (4-wire is automatically assumed if 3-wire is false).

B) Specify and define if 16-bit or 8-bit data organization is used, by setting equate 'org8' TRUE or FALSE.

C) The user should assemble the source file by specifying which type of serial EEPROM is being used.

FIGURE 1A - 4 WIRE CONNECTION

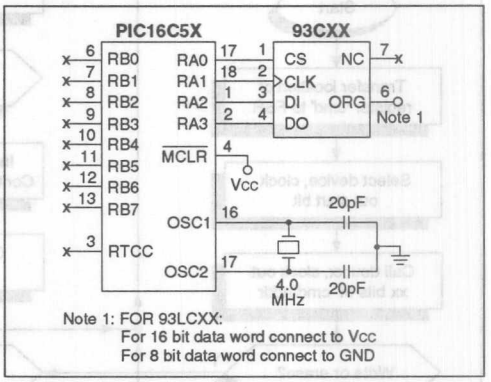
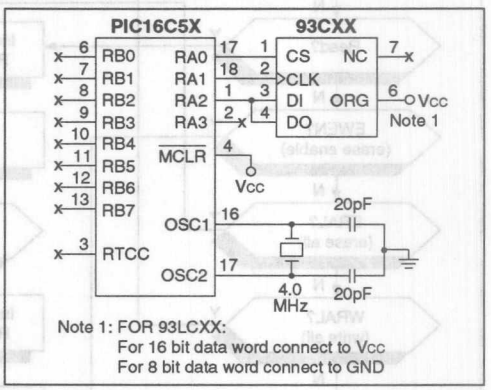


FIGURE 1B - 3 WIRE CONNECTION



This is done by defining the equate S93C46, S93LC46, etc., as TRUE. Only one device can be TRUE, the rest have to be defined FALSE.

D) The user would invoke the driver as follows:

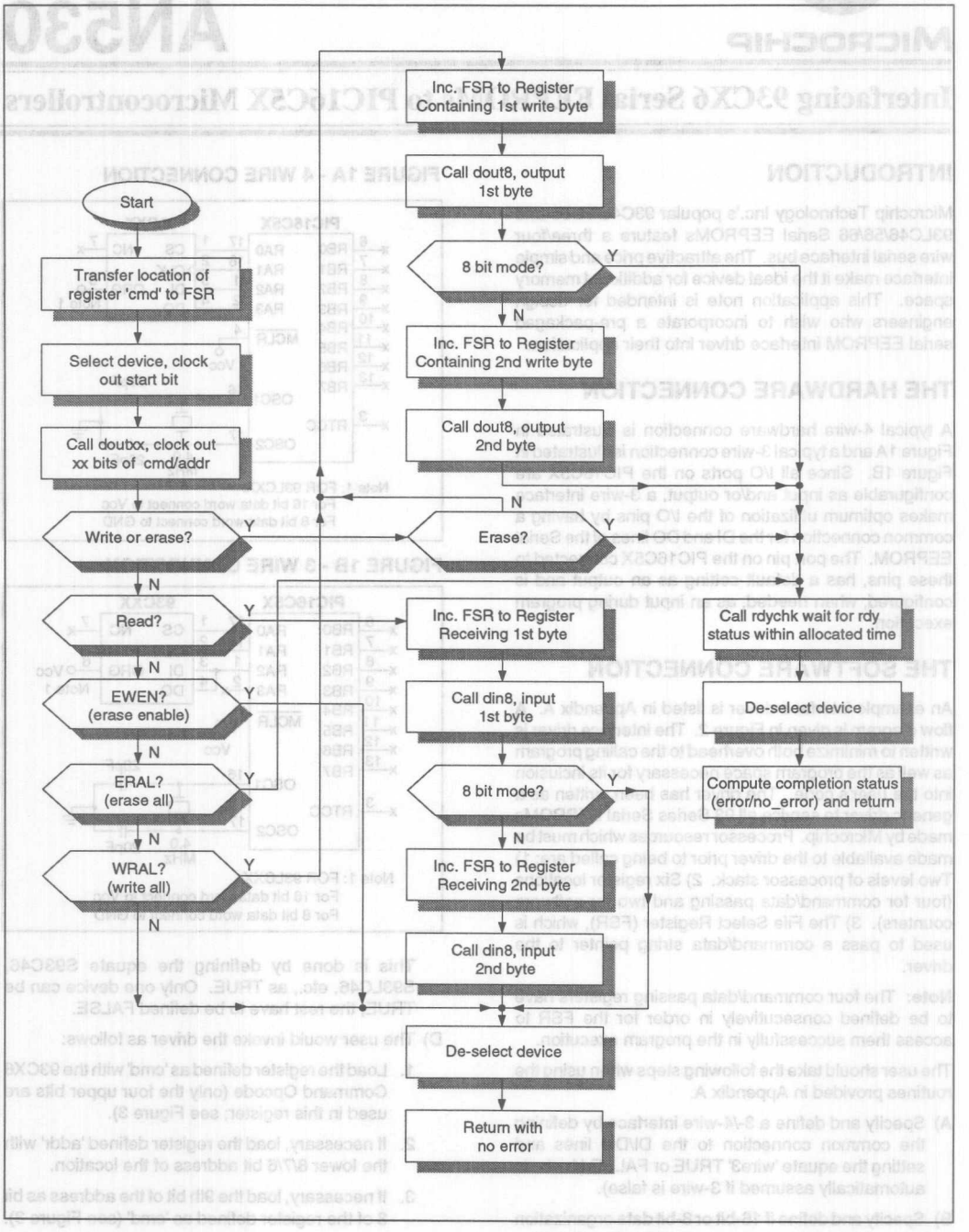
1. Load the register defined as 'cmd' with the 93CX6 Command Opcode (only the four upper bits are used in this register; see Figure 3).
2. If necessary, load the register defined 'addr' with the lower 8/7/6 bit address of the location.
3. If necessary, load the 9th bit of the address as bit 3 of the register defined as 'cmd' (see Figure 3).

   Note: READ, WRITE and ERASE commands need to have an address associated with the command and the 9th bit of the address is only

7

© 1993 Microchip Technology Incorporated

FIGURE 2 - FLOW CHART



© 1993 Microchip Technology Incorporated

required when using the 93C56/66 or 93LC56/66 devices in the 8 bit mode (ORG tied to GND).

4. If necessary, load the register defined as 'highb' and 'lowb' with the 16 bits of data, the most significant byte loaded in 'highb'. In 8 bit mode, 'highb' should be loaded with the 8 bit data.

   Note: Only the WRITE and WRAL commands require data to be loaded in the 'highb', 'lowb' locations.

5. Call the driver sub-routine 'see'.
6. Upon completion, the driver will return a completion status in the W register (error/no error). Only commands requiring a status check are capable of returning a valid error/no error status, in all other cases a no error is returned.
7. If the READ command is executed, the 16/8 bit data will be loaded in the 'highb' and 'lowb' registers, where 'highb' contains the MSB in the 16 bit mode and 8 bit data in the 8-bit mode.

   The Example interface assumes a 4 MHz oscillator clock which gives us a 1 µS instruction cycle. If a higher clock speed is used, additional NOPs have to be included in the code in order to meet the minimum clock speed requirements of the 93 Series Serial EEPROMs (see data sheet for further details).

Listing in Appendix B is for an interface to 93C46 Serial EEPROM only.

SUMMARY

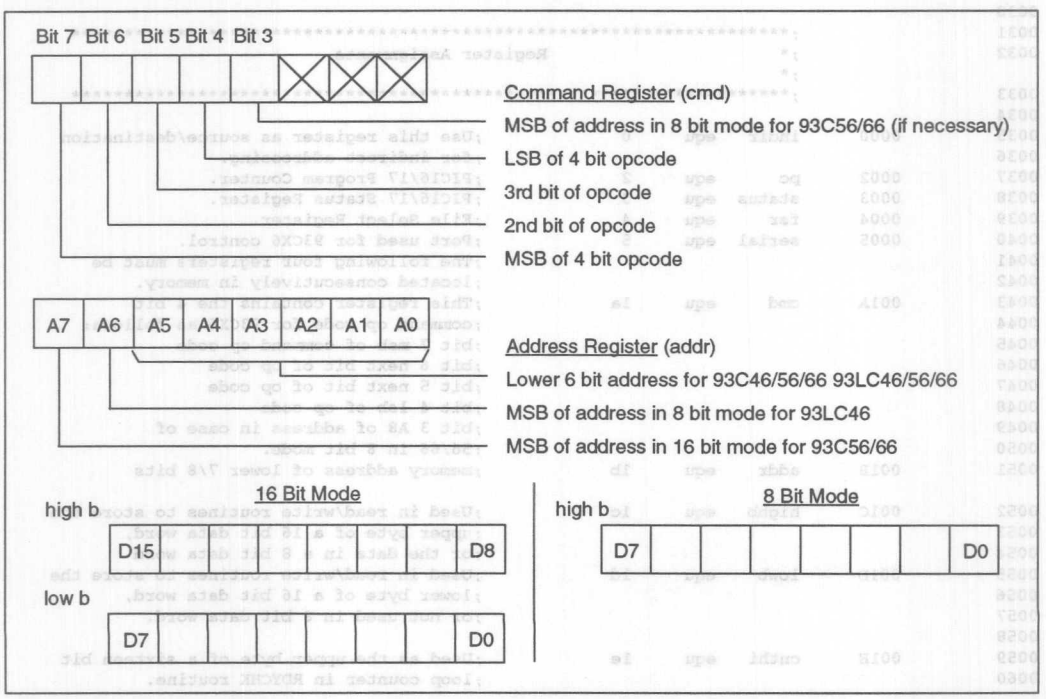
The 93 Series Serial EEPROMs are a simple and versatile method of increasing read/write memory capability in a PIC16C5X application. The 'generic' code in Appendix A makes it easy to incorporate in any PIC16C5X application. Any of the 93 Series Serial EEPROMs can be applied, while at the same time using a minimal amount of I/O, code and RAM resources.

Code size: 6 bytes of RAM.

Appendix A listing:	127 bytes program code (max.)
	100 bytes program code (min.)
Appendix B listing:	86 bytes program code

Authors: *Stan D'Souza*
Logic Products Division
Bob Ward
Field Applications/Sales Organization

FIGURE 3 - 'CMD', 'ADDR', DATA BYTE DEFINITION



7

Appendix A

```
Microchip Technology 16c5X Assembler 3.03A   Wed Feb 26 11:09:07 1992  Page 1
R/W EEPROM

Line   PC    Opcode

0001              LIST  P = 16C54
0002                                ;
0003                                ;Define Equates:
0004                                ;
0005  01FF        PIC54   EQU   1FFH
0006                                ;
0007                                ;
0008  0000              ORG   0
0009  START
0010   0000   0A7B   goto    main          ;run test program
0011                                ;
0012                                ;
0013  0001      TRUE    EQU     1
0014  0000      FALSE   EQU     0
0015  0000      S93C46  EQU     FALSE
0016  0000      S93LC46 EQU     FALSE
0017  0000      S93C56  EQU     FALSE
0018  0000      S93LC56 EQU     FALSE
0019  0001      S93C66  EQU     TRUE
0020  0000      S93LC66 EQU     FALSE
0021  0001      wire3   equ     TRUE          ;for four-wire setup equate to FALSE
0022  0000      org8    EQU     FALSE
0023                                ;
0024  0001      H16     EQU     TRUE
0025  0000      H8      EQU     FALSE
0026  0000      LC468   EQU     FALSE
0027  0000      XC46    EQU     FALSE
0028
0029
0030
0031                ;*****************************************************************
0032                ;*                      Register Assignments
                    ;*
0033                ;*****************************************************************
0034
0035  0000      indir   equ     0             ;Use this register as source/destination
0036                                ;for indirect addressing.
0037  0002      pc      equ     2             ;PIC16/17 Program Counter.
0038  0003      status  equ     3             ;PIC16/17 Status Register.
0039  0004      fsr     equ     4             ;File Select Register.
0040  0005      serial  equ     5             ;Port used for 93CX6 control.
0041                                ;The following four registers must be
0042                                ;located consecutively in memory.
0043  001A      cmd     equ     1a            ;This register contains the 4 bit
0044                                ;command op code for 93CX6 as follows:
0045                                ;bit 7 msb of command op code
0046                                ;bit 6 next bit of op code
0047                                ;bit 5 next bit of op code
0048                                ;bit 4 lsb of op code
0049                                ;bit 3 A8 of address in case of
0050                                ;56/66 in 8 bit mode.
0051  001B      addr    equ     1b            ;memory address of lower 7/8 bits

0052  001C      highb   equ     1c            ;Used in read/write routines to store the
0053                                ;upper byte of a 16 bit data word,
0054                                ;or the data in a 8 bit data word
0055  001D      lowb    equ     1d            ;Used in read/write routines to store the
0056                                ;lower byte of a 16 bit data word,
0057                                ;or not used in 8 bit data word.
0058
0059  001E      cnthi   equ     1e            ;Used as the upper byte of a sixteen bit
0060                                ;loop counter in RDYCHK routine.
```


© 1993 Microchip Technology Incorporated

```
0061        001F    cnt     equ     1f          ;Used as the lower byte of a sixteen bit
0062                                            ;loop counter in RDYCHK routine, and
0063                                            ;elswhere as an eight bit counter.
0064        001E    temp_cmd  equ     1e        ;doubles as a temp register for cmd
0065        001F    temp_addr equ     1f        ;doubles as a temp register for addr
0066
0067                ;*****************************************************************
0068                ;*                     Bit Assignments
                    ;* The following assignments are for 3-wire.  For 4-wire please assign
0069                ;* din and dout to two separate pins.
0070
0071        0000    carry   equ     0           ;Carry Flag of Status Register.
0072        0002    zflag   equ     2           ;Zero Flag of Status Register.
0073
0074        0002    cs      equ     2           ;Port pin tied to CS on 93CX6.
0075        0001    din     equ     1           ;Port pin tied to DI on 93CX6.  3-wire setup
0076        0001    dout    equ     1           ;Port pin tied to DO on 93CX6.  3-wire setup
0077        0003    clock   equ     3           ;Port pin tied to CLK on 93CX6.
0078
0079                ;*****************************************************************
0080                ;*                   General Assignments
                    ;*
0081                ;*****************************************************************
0082
0083        0000    no_err  equ     0
0084        0001    error   equ     1
0085        0020    tries   equ     20          ;After issuing a WRITE, ERASE, ERAL, or WRAL
0086                                            ;command, the approximate number of machine
0087                                            ;cycles X 256 to wait for the RDY status.
0088                                            ;This value must be adjusted for operating
0089                                            ;frequencies other than 4 MHz.
0090
0091        0080    read    equ     b'10000000' ;read command op code
0092        0040    write   equ     b'01000000' ;write command op code
0093        00C0    erase   equ     b'11000000' ;erase command op code
0094        0030    ewen    equ     b'00110000' ;erase enable command opcode
0095        0000    ewds    equ     b'00000000' ;erase disable command opcode
0096        0020    eral    equ     b'00100000' ;erase all command op code
0097        0010    wral    equ     b'00010000' ;write all command op code
0098
0099                ;*****************************************************************
0100                ;*                   Macro Definitions
                    ;*
0101                ;*****************************************************************
0102
0103                sel     MACRO               ;Selects the 93CX6 device
0104                bsf     serial,cs           ;Chip Select (CS) = '1' to select
0105                ENDM                        ;the device
0106
0107                dsel    MACRO               ;De-select the 93CX6 device.
0108                bcf     serial,cs           ;Chip Select (CS) = '0' to de-select
0109                                            ;the device.
0110                ENDM
0111
0112                strtbt  MACRO               ;Issue the Start Bit to the 93CX6.
0113                bsf     serial,din          ;Start Bit = '1'.
0114                clkit                       ;Clock it out.
0115                ENDM
0116
0117                clkit   MACRO               ;Clocks a serial data bit into or out
0118                                            ;of the 93CX6 device.
0119                bsf     serial,clock        ;Clock (CLK) = '1'.
0120
0121                nop                         ;Adjust the number of nop instructions
0122                                            ;between the assertion and de-assertion of
0123                                            ;CLK in proportion to the PIC16/17 operating
0124                                            ;frequency.  Refer to the 93CX6 data for the
0125                                            ;minimum CLK period.
0126
```

7

© 1993 Microchip Technology Incorporated

```
0127                    bcf     serial,clock            ;Clock (CLK) = '0'.
0128                    ENDM
0129                    ;
0130                    ;*********************************************************************
0131                    ;*                              DOUTx                                *
0132                    ;*********************************************************************
0133                    ;doutxx, outputs up to 11 bits of op code/data, depending on whether
0134                    ;a 46/56/66 serial eeprom is being used. The number of bits over 8 are
0135                    ;saved in the cmd register and the rest in the addr register. Before
0136                    ;calling this routine the cmd and the addr registers should be loaded
0137                    ;as follows:
0138                    ;cmd reg.bits 7|6|5|4|3|2|1|0
0139                    ;-------------|-|-|-|-|-|-|-|
0140                    ;             X|X|X|X|X|X|X|Y| -> not used
0141                    ;             X|X|X|X|X|X|Y|X| -> mot used
0142                    ;             X|X|X|X|X|Y|X|X| -> not used
0143                    ;             X|X|X|X|Y|X|X|X| -> 9th bit of address if necessary
0144                    ;             X|X|X|Y|X|X|X|X| -> lsb of command op code
0145                    ;             X|X|Y|X|X|X|X|X| -> 3rd bit of command opcode
0146                    ;             X|Y|X|X|X|X|X|X| -> 2nd bit of command opcode
0147                    ;             Y|X|X|X|X|X|X|X| -> msb of command op code
0148                    ;
0149                    ;addr reg. 6/7/8 bits of address if necessary.
0150
0151                    dout10
0152  0001  0360                rlf     indir           ;rotate thru carry
0153  0002  0425                bcf     serial,din      ;set output to 0
0154  0003  0603                btfsc   status,carry    ;set?
0155  0004  0525                bsf     serial,din      ;else set output to 1
0156                            clkit                   ;clk data
0157                    m
0158                    m                               ;of the 93CX6 device.
0159  0005  0565        m       bsf     serial,clock    ;Clock (CLK) = '1'.
0160                    m
0161  0006  0000        m       nop                     ;Adjust the number of nop instructions
0162                    m                               ;between the assertion and de-assertion of
0163                    m                               ;CLK in proportion to the PIC16/17 operating
0164                    m                               ;frequency.  Refer to the 93CX6 data for the
0165                    m                               ;minimum CLK period.
0166                    m
0167  0007  0465                bcf     serial,clock    ;Clock (CLK) = '0'.
0168                    dout9
0169  0008  0360                rlf     indir           ;rotate thru carry
0170  0009  0425                bcf     serial,din      ;set output to 0
0171  000A  0603                btfsc   status,carry    ;set?
0172  000B  0525                bsf     serial,din      ;else set output to 1
0173                            clkit                   ;clk data
0174                    m
0175                    m                               ;of the 93CX6 device.
0176  000C  0565        m       bsf     serial,clock    ;Clock (CLK) = '1'.
0177                    m
0178  000D  0000        m       nop                     ;Adjust the number of nop instructions
0179                    m                               ;between the assertion and de-assertion of
0180                    m                               ;CLK in proportion to the PIC16/17 operating
0181                    m                               ;frequency.  Refer to the 93CX6 data for the
0182                    m                               ;minimum CLK period.
0183                    m
0184  000E  0465                bcf     serial,clock    ;Clock (CLK) = '0'.
0185  000F  02A4                incf    fsr             ;inc pointer
0186                    dout8
0187  0010  0C08                movlw   8               ;Initialize loop counter.
0188  0011  003F                movwf   cnt
0189                                                    ;
0190  0012  0425        d_o_8   bcf     serial,din      ;Assume that the bit to be transfered is a
0191                                                    ;'0'.  Hence, de-assert DI.
0192  0013  0360                rlf     indir           ;Rotate the actual bit to be transferred into
0193                                                    ;the carry bit.
0194  0014  0603                btfsc   status,carry    ;Test the carry, if our assumption was
0195                                                    ;correct, skip the next instruction.
```


© 1993 Microchip Technology Incorporated

```
0196  0015  0525          bsf     serial,din      ;No, actual bit was a '1'. Assert DI.
0197                      clkit                   ;Clock the 93CX6.
0198                m
0199                m                             ;of the 93CX6 device.
0200  0016  0565    m     bsf     serial,clock    ;Clock (CLK) = '1'.
0201                m
0202  0017  0000    m     nop                     ;Adjust the number of nop instructions
0203                m                             ;between the assertion and de-assertion of

0204                m                             ;CLK in proportion to the PIC16/17 operating
0205                m                             ;frequency.  Refer to the 93CX6 data for the
0206                m                             ;minimum CLK period.
0207                m
0208  0018  0465          bcf     serial,clock    ;Clock (CLK) = '0'.
0209  0019  02FF          decfsz  cnt             ;Repeat until cnt = 0.
0210  001A  0A12          goto    d_o_8           ;Cnt still > 0.
0211  001B  0360          rlf     indir           ;Restore register to its original condition.
0212  001C  0425          bcf     serial,din      ;make sure din is low
0213  001D  0800          retlw   no_err          ;Exit with good status.
0214
0215                ;*****************************************************************
0216                ;*                          DIN8
                    ;*
0217                ;*****************************************************************
0218                                              ;Din8 will input 8 bits of data from the
0219                                              ;93CX6.  Before calling this routine, the FSR
0220                                              ;must point to the register being used to
0221                                              ;hold the incomming data.
0222                din8
0223                                              ;set up the RA1 as a input before proceeding
0224  001E  0C02          movlw   b'00000010'     ;set up porta
0225  001F  0005          tris    serial          ;
0226  0020  0C08          movlw   8               ;Initialize loop counter.
0227  0021  003F          movwf   cnt             ;
0228
0229                d_i_8
0230  0022  0360          rlf     indir           ;Make room for the incomming bit in the
0231                                              ;destination register.
0232  0023  0400          bcf     indir,0         ;Assume that the incomming bit is a '0' and
0233                                              ;clear the LSB of the destination register.
0234                      clkit                   ;Clock a bit in the 93CX6.
0235                m
0236                m                             ;of the 93CX6 device.
0237  0024  0565    m     bsf     serial,clock    ;Clock (CLK) = '1'.
0238                m
0239  0025  0000    m     nop                     ;Adjust the number of nop instructions
0240                m                             ;between the assertion and de-assertion of
0241                m                             ;CLK in proportion to the PIC16/17 operating
0242                m                             ;frequency.  Refer to the 93CX6 data for the
0243                m                             ;minimum CLK period.
0244                m
0245  0026  0465          bcf     serial,clock    ;Clock (CLK) = '0'.
0246  0027  0625          btfsc   serial,dout     ;Test the incomming bit, if our assumption
0247                                              ;was correct, skip the next instruction.
0248  0028  0500          bsf     indir,0         ;No, actual bit is a '1'.  Set the LSB of the
0249                                              ;destination register.
0250  0029  02FF          decfsz  cnt             ;Repeat until cnt = 0.
0251  002A  0A22          goto    d_i_8           ;Cnt still > 0.
0252                                              ;setup RA1 back to output
0253  002B  0C00          movlw   0               ;set RA1 as output
0254  002C  0005          tris    serial          ;      /
0255  002D  0800          retlw   no_err          ;Exit with good status.
0256
0257                ;*****************************************************************
0258                ;*                          RDYCHK
                    ;*
0259                ;*****************************************************************
0260                                              ;Rdychk will read the 93CX6 READY/BUSY status
0261                                              ;and wait for RDY status within the alloted
0262                                              ;number of processor cycles.  If RDY status
```


© 1993 Microchip Technology Incorporated

```
0263                                        ;is not present after this set period, the
0264                                        ;routine will return with an error status.
0265
0266                    rdychk
0267                                        ;set up RA1 as a input before proceeding
0268    002E  0C02             movlw   b'00000010'  ;set up porta
0269    002F  0005             tris    serial       ;
0270    0030  0C20             movlw   tries        ;Initialize time-out counter
0271    0031  003E             movwf   cnthi        ;
0272    0032  007F             clrf    cnt          ;
0273                           dsel                 ;De-select the 93CX6.
0274                    m
0275    0033  0445     m       bcf     serial,cs    ;Chip Select (CS) = '0' to de-select
0276                                                ;the device.
0277
0278                    ;      nop                  ;NOTE:  Check the 93CX6 data sheet for
0279                                                ;minimum CS low time.  Depending upon
0280                                                ;processor frequency, a nop(s) may be
0281                                                ;between the assertion and de-assertion of
0282                                                ;Chip Select.
0283
0284                           sel                  ;Re-select the 93CX6.
0285                    m
0286    0034  0545             bsf     serial,cs    ;Chip Select (CS) = '1' to select
0287    0035  0625     notrdy  btfsc   serial,dout  ;If DO is a '0', 93CX6 has yet to completed
0288                                                ;the last operation (still busy).
0289    0036  0A3E             goto    rdynoerr             ;skip to no error
0290    0037  02FF             decfsz  cnt          ;No, not yet ready.  Decrement the LSB of
our
0291                                                ;16 bit timer and check for expiration.
0292    0038  0A35             goto    notrdy       ;Still some time left.  Try again.
0293    0039  02FE             decfsz  cnthi        ;Least significant byte expired - decrement
0294                                                ;and check for expiration of the MSB.
0295    003A  0A35             goto    notrdy       ;Still some time left.  Try again.
0296                                                ;setup RA1 back to output
0297    003B  0C00             movlw   0            ;set RA1 as output
0298    003C  0005             tris    serial       ;      /
0299    003D  0801             retlw   error        ;RDY status was not present in the alloted
0300                                                ;time, return with error status.
0301                    rdynoerr
0302                                                ;setup RA1 back to output
0303    003E  0C00             movlw   0            ;set porta as output
0304    003F  0005             tris    serial       ;      /
0305    0040  0800             retlw   no_err
0306
0307                    ;*****************************************************************
0308                    ;*                          SEE
                        ;*
0309                    ;*****************************************************************
0310
0311                                                ;See will control the entire operation of a
0312                                                ;93CX6 device. Prior to calling the routine,
0313                                                ;load a valid command/memory address into
0314                                                ;location cmd, and for WRITE or WRAL
0315                                                ;commands, load registers highb and lowb with
0316                                                ;16 bits of write data.  Upon exit, the W
0317                                                ;register will contain the completion status.
0318                                                ;Only 93CX6 instructions which require a
0319                                                ;status check can return with an error as the
0320                                                ;completion status.  The values that denote
0321                                                ;the completion status are defined as
0322                                                ;variables 'error' and 'no_err' in the
0323                                                ;general assignments section.
0324
0325                    see
0326    0041  021A             movf    cmd,w        ;save cmd
0327    0042  003E             movwf   temp_cmd
0328    0043  021B             movf    addr,w       ;save addr
0329    0044  003F             movwf   temp_addr    ;
0330    0045  0C1A             movlw   cmd          ;Load W with the location of the cmd
0331                                                ;register.
```


© 1993 Microchip Technology Incorporated

```
0332   0046  0024          movwf   fsr             ;Transfer that information into the File
0333                                                ;Select Register.  The fsr now points to
0334                                                ;location cmd.
0335                       sel                      ;Select the 93CX6.
0336                 m
0337   0047  0545          bsf     serial,cs        ;Chip Select (CS) = '1' to select
0338                       strtbt                   ;Send a start bit.
0339                 m
0340   0048  0525          bsf     serial,din       ;Start Bit = '1'.
0341                       clkit                    ;Clock it out.
0342                 m
0343                 m                              ;of the 93CX6 device.
0344   0049  0565          bsf     serial,clock     ;Clock (CLK) = '1'.
0345                 m
0346   004A  0000          nop                      ;Adjust the number of nop instructions
0347                 m                              ;between the assertion and de-assertion of
0348                 m                              ;CLK in proportion to the PIC16/17 operating
0349                 m                              ;frequency.  Refer to the 93CX6 data for the
0350                 m                              ;minimum CLK period.
0351                 m
0352   004B  0465          bcf     serial,clock     ;Clock (CLK) = '0'.
0353                 ;
0354   004C  06FA          btfsc   cmd,7            ;bit 7 = 0?
0355   004D  0A73          goto    sca10            ;xfer 10 bit cmd/adr
0356
0357                 ;     goto    set_cmd_addr     ;no then set cmd/addr
0358   004E  07DA          btfss   cmd,6            ;bit 6 = 0 ?
0359   004F  0A75          goto    sc10             ;xfer 10 bit cmd/adr
0360                 ;     goto    set_cmd          ;yes then set cmd
0361   0050  0A73          goto    sca10            ;xfer 10 bit cmd/adr
0362
0363                 ;     goto    set_cmd_addr     ;else set cmd/addr
0364                 see1
0365   0051  021E          movf    temp_cmd,w       ;retore cmd
0366   0052  003A          movwf   cmd              ;      /
0367   0053  021F          movf    temp_addr,w      ;restore addr
0368   0054  003B          movwf   addr             ;      /
0369   0055  06DA          btfsc   cmd,6            ;Check for a WRITE or ERASE command.
0370   0056  0A5F          goto    see2             ;Yes, parse the command further.
0371   0057  06FA          btfsc   cmd,7            ;Check for a READ command.
0372   0058  0A69          goto    read_            ;Yes, process READ command.
0373   0059  06BA          btfsc   cmd,5            ;Check for a EWEN or ERAL command.
0374   005A  0A66          goto    see3             ;Yes, parse the command further.
0375   005B  069A          btfsc   cmd,4            ;Check for a WRAL command.
0376   005C  0A6E          goto    write_           ;Yes, process WRITE/WRAL command.
0377
0378                 exit_ dsel                     ;No further processing required; 93CX6
0379                 m
0380   005D  0445    m     bcf     serial,cs        ;Chip Select (CS) = '0' to de-select
0381                                                ;the device.
0382                                                ;command completed.
0383                 ;
0384   005E  0800          retlw   no_err           ;Return with good completion status.
0385
0386   005F  07FA    see2  btfss   cmd,7            ;Check for a ERASE command.
0387   0060  0A6E          goto    write_           ;No, process WRITE command.
0388   0061  092E    exit2_ call   rdychk           ;ERASE command requires a status check.
0389                       dsel                     ;De-select the 93CX6.
0390                 m
0391   0062  0445    m     bcf     serial,cs        ;Chip Select (CS) = '0' to de-select
0392                                                ;the device.
0393   0063  01E2          addwf   pc               ;Compute completion status from results of
0394                                                ;status check.
0395   0064  0800          retlw   no_err           ;Return with good completion status.
0396   0065  0801          retlw   error            ;Return with bad completion status.
0397
0398   0066  069A    see3  btfsc   cmd,4            ;Check for a EWEN command
0399   0067  0A5D          goto    exit_            ;Yes, no further processing required, exit
0400                                                ;now.
0401   0068  0A61          goto    exit2_           ;No, ERAL command which requires a status
```


© 1993 Microchip Technology Incorporated

```
0402                                                    ;check.
0403
0404   0069  02A4    read_   incf    fsr               ;Increment the File Select Register to point
0405                                                    ;to the register receiving the upper byte of
0406                                                    ;the incomming 93CX6 data word.
0407   006A  091E            call    din8              ;Input the upper byte.
0408   006B  02A4            incf    fsr               ;Increment the File Select Register to point
0409                                                    ;to the register receiving the lower byte.
0410   006C  091E            call    din8              ;Input 8 more bits.
0411   006D  0A5D            goto    exit_             ;No further processing required, exit now.
0412
0413   006E  02A4    write_  incf    fsr               ;Increment the File Select Register to point
0414                                                    ;to the upper byte of the 16 bit 93CX6 data
0415                                                    ;word to be transmitted.
0416   006F  0910            call    dout8             ;Output that byte.
0417   0070  02A4            incf    fsr               ;Increment the File Select Register to point
0418                                                    ;to the lower byte.
0419   0071  0910            call    dout8             ;Output the lower byte of the 16 bit 93CX6
0420                                                    ;data word.
0421   0072  0A61            goto    exit2_            ;Exit with a status check.
0422
0423                 ;
0424                 ;
0425                 ;
0426                 sca10
0427   0073  0901            call    dout10            ;output cmd reg
0428   0074  0A51            goto    see1              ;return
0429                 ;
0430                 ;
0431                 ;
0432                 ;
0433                 ;
0434                 sc10
0435   0075  037A            rlf     cmd               ;rotate cmd
0436   0076  035A            rlf     cmd,w             ;left twice
0437   0077  003B            movwf   addr              ;save in addr
0438   0078  007A            clrf    cmd               ;clear command
0439   0079  0901            call    dout10            ;xfer 10 bits
0440   007A  0A51            goto    see1              ;return
0441                 ;
0442                 ;*****************************************************************
0443                 ;*                          Test Program                         *
                     ;*                                                               *
0444                 ;*****************************************************************
0445         main                                       ;We've include a sample program to exercise
0446                                                    ;the PIC16/17 to 93C66 interface using a simple
0447                                                    ;erase, write and verify routine.
0448                                                    ;8 bit organization has been used
0449                                                    ;with a 3 wire interface.
0450
0451   007B  0065            clrf    serial            ;Clear the port tied to the 93C66 device.
0452   007C  0CF0            movlw   b'11110000'       ;Intialize the data direction register for
0453   007D  0005            tris    serial            ;that port.
0454
0455   007E  0C30            movlw   ewen              ;Load W with the Erase/Write Enable command.
0456   007F  003A            movwf   cmd               ;Transfer W into cmd register.
0457   0080  0941            call    see               ;Enable the 93C66 device.
0458
0459   0081  0C20            movlw   eral              ;Load W with the Erase All command.
0460   0082  003A            movwf   cmd               ;Transfer W into cmd register.
0461   0083  0941            call    see               ;Erase the 93C66.
0462   0084  0F01            xorlw   error             ;Check completion status.
0463   0085  0643            btfsc   status, zflag     ;Test for error condition.
0464   0086  0AA6            goto    errloop           ;Yes, bad completion status, error-out.
0465
0466                                                    ;Write loop:
0467   00AA      tstptrn     equ     0xAA              ;Define the test pattern to be written.
0468
0469   0087  0C40            movlw   write             ;Load W with the Write command.
0470   0088  003A            movwf   cmd               ;Transfer W into cmd register.
```


© 1993 Microchip Technology Incorporated

```
0471   0089  0CAA            movlw   tstptrn         ;Intialize the 93C66 data registers with
0472                                                  ;write data.
0473   008A  003C            movwf   highb           ;load in high byte only
0474                                                  ;since 8 bit low byte is ignored
0475   008B  007B            clrf    addr            ;start at addr 0
0476   008C  0941  test1     call    see             ;Write data word into 93C66 device.
0477   008D  0F01            xorlw   error           ;Check completion status.
0478   008E  0643            btfsc   status,zflag    ;Test for error condition.
0479   008F  0AA6            goto    errloop         ;Yes, bad completion status, error-out.
0480   0090  03FB            incfsz  addr            ;No, increment the 8 bit memory address
0481                                                  ;field.
0482   0091  0A8C            goto    test1           ;write another location
0483   0092  077A            btfss   cmd,3           ;see if all done
0484   0093  0A95            goto    wrt_nxt_pg      ;no then write next page
0485   0094  0A97            goto    read_tst        ;read written data
0486               wrt_nxt_pg
0487   0095  057A            bsf     cmd,3           ;set page bit
0488   0096  0A8C            goto    test1           ;No, write another location.

0489
0490                                                  ;Read loop:
0491               read_tst
0492   0097  0C80            movlw   read            ;Load W with the Read command.
0493   0098  003A            movwf   cmd             ;Transfer W into cmd register.
0494   0099  0941  test2     call    see             ;Read addressed data word from 93C66 device.
0495   009A  0CAA            movlw   tstptrn         ;Load W with the pattern written.
0496   009B  009C            subwf   highb,w         ;Verify the data read against what was
0497                                                  ;written.
0498   009C  0743            btfss   status,zflag    ;Same?
0499   009D  0AA6            goto    errloop         ;No, error-out.
0500   009E  03FB            incfsz  addr            ;Yes, both byte correct, increment the 8 bit
0501                                                  ;memory address field.
0502   009F  0A99            goto    test2           ;do next byte
0503   00A0  077A            btfss   cmd,3           ;check page bit
0504   00A1  0AA3            goto    rd_nxt_pg       ;no then chk next page
0505   00A2  0AA5            goto    allok           ;all done!!!
0506               rd_nxt_pg
0507   00A3  057A            bsf     cmd,3           ;set page bit
0508   00A4  0A99            goto    test2           ;No, read another location.
0509
0510   00A5  0AA5  allok     goto    allok           ;Home safe!
0511               ;
0512               ;
0513               errloop
0514   00A6  0AA6            goto    errloop
0515               ;
0516               ;
0517               ;
0518               ;KEY DEFINITIONS
0519               ;
0520         0000            ORG     PIC54
0521               SYS_RESET
0522   01FF  0A00            GOTO    START
0523               ;
0524         0000      END
```


© 1993 Microchip Technology Incorporated

Appendix B

```
Microchip Technology PIC16C5X Assembler 3.03A   Wed Feb 26 11:11:15 1992  Page 1

Line   PC    Opcode

0001                ;*********************************************************************
0002                ;*                    MPALC Directives Section                       *
                    ;*                                                                   *
0003                ;*********************************************************************
0004
0005                    LIST     P=16C54,N=40,C=132
0006
0007
0008                    ;*********************************************************************
0009                    ;*                    Register Assignments                           *
0010                    ;*********************************************************************
0011
0012        0000  indir   equ     0x00      ;Use this register as source/destination for
0013                                        ;indirect addressing.
0014        0002  pc      equ     0x02      ;PIC16/17 Program Counter.
0015        0003  status  equ     0x03      ;PIC16/17 Status Register.
0016        0004  fsr     equ     0x04      ;File Select Register.
0017        0005  serial  equ     0x05      ;Port used for 93C46 control.  Since Port A
0018                                        ;is 4 bits wide, we'll use 3 or 4 pins of Port A.
0019
0020                                        ;The following three registers must be
0021                                        ;located consecutively in memory.
0022        0010  cmd     equ     0x10      ;This register contains the 2 bit 93C46
0023                                        ;command is the upper 2 bit positions and
0024                                        ;memory address in the lower 6.
0025        0011  highb   equ     0x11      ;Used in read/write routines to store the
0026                                        ;upper byte of a 16 bit 93C46 data word.
0027        0012  lowb    equ     0x12      ;Used in read/write routines to store the
0028                                        ;lower byte of a 16 bit 93C46 data word.
0029
0030        0013  cnthi   equ     0x13      ;Used as the upper byte of a sixteen bit loop
0031                                        ;counter in RDYCHK routine.
0032        0014  cnt     equ     0x14      ;Used as the lower byte of a sixteen bit loop
0033                                        ;counter in RDYCHK routine, and elsewhere as
0034                                        ;an eight bit counter.
0035
0036                    ;*********************************************************************
0037                    ;*  Bit Assignments:  The following assignment is for 3-wire setup.
0038                    ;*  For four wire setup, assign dout equ 2.
0039
0040        0000  carry   equ     0         ;Carry Flag of Status Register.
0041        0002  zflag   equ     2         ;Zero Flag of Status Register.
0042
0043        0000  cs      equ     0         ;Port pin tied to CS on 93C46.
0044        0001  din     equ     1         ;Port pin tied to DI on 93C46.  3-wire setup.
0045        0001  dout    equ     1         ;Port pin tied to DO on 93C46.  3-wire setup.
0046        0003  clock   equ     3         ;Port pin tied to CLK on 93C46.
0047
0048                    ;*********************************************************************
0049                    ;*                  General Assignments                              *
0050                    ;*********************************************************************
0051
0052        0000  no_err  equ     0         ;
0053        0001  error   equ     1         ;
0054        0004  tries   equ     0x04      ;After issuing a WRITE, ERASE, ERAL, or WRAL
0055                                        ;command, the approximate number of machine
0056                                        ;cycles X 256 to wait for the RDY status.
0057                                        ;This value must be adjusted for operating
0058                                        ;frequencies other than 4 MHz.
0059
0060        0080  read    equ     0x80      ;93C46 Read command.
0061        0040  write   equ     0x40      ;93C46 Write command.
0062        00C0  erase   equ     0xC0      ;93C46 Erase command.
0063             0030  ewen    equ     0x30 ;93C46 Erase/Write Enable command.
```


© 1993 Microchip Technology Incorporated

```
0064  0000        ewds    equ     0x00            ;93C46 Erase/Write Disable command.
0065  0020        eral    equ     0x20            ;92CXX Erase All command.
0066  0010        wral    equ     0x10            ;92CXX Write All command.
0067
0068                      ;*****************************************************************
0069                      ;*                       Macro Definitions                       *
0070                      ;*****************************************************************
0071
0072          sel         MACRO                   ;Selects the 93C46 device.
0073                      bsf     serial,cs       ;Chip Select (CS) = '1' to select the device.
0074                      ENDM
0075
0076          dsel        MACRO                   ;De-select the 93C46 device.
0077                      bcf     serial,cs       ;Chip Select (CS) = '0' to de-select the
0078                                              ;device.
0079                      ENDM
0080
0081                      strtbt  MACRO           ;Issue the Start Bit to the 93C46.
0082                      bsf     serial,din      ;Start Bit = '1'.
0083                      clkit                   ;Clock it out.
0084                      ENDM
0085
0086                      clkit   MACRO           ;Clocks a serial data bit into or out of the
0087                                              ;93C46 device.
0088                      bsf     serial,clock    ;Clock (CLK) = '1'.
0089
0090                      nop                     ;Adjust the number of nop instructions
0091                                              ;between the assertion and de-assertion of
0092                                              ;CLK in proportion to the PIC16/17 operating
0093                                              ;frequency.  Refer to the 93C46 data for the
0094                                              ;minimum CLK period.
0095
0096                      bcf     serial,clock    ;Clock (CLK) = '0'.
0097                      ENDM
0098
0099                      ;*****************************************************************
0100                      ;*                   Power-On/Reset Entry Point                  *
0101                      ;*****************************************************************
0102
0103          0000  reset_ org     0x1FF
0104  01FF    0A57        goto    main
0105
0106                      ;*****************************************************************
0107                      ;*                       93C46 Routines                          *
0108                      ;*****************************************************************
0109          0000        org     0x000           ;Locate all subroutines in the lower half of
0110                                              ;a Program Memory Page.
0111
0112                      ;*****************************************************************
0113                      ;*                          DOUT8                                *
0114                      ;*****************************************************************
0115                                              ;Dout8 will output 8 bits of data to the
0116                                              ;93C46.  Before calling this routine, the FSR
0117                                              ;must point to the byte being transmitted.
0118
0119  0000  0C08  dout8   movlw   0x08            ;Initialize loop counter.
0120  0001  0034         movwf   cnt             ;
0121
0122  0002  0425  d_o_8   bcf     serial,din      ;Assume that the bit to be transfered is a
0123                                              ;'0'.  Hence, de-assert DI.
0124  0003  0360         rlf     indir           ;Rotate the actual bit to be transferred into
0125                                              ;the carry bit.
0126  0004  0603         btfsc   status,carry;   Test the carry, if our assumption was
0127                                              ;correct, skip the next instruction.
0128  0005  0525         bsf     serial,din      ;No, actual bit was a '1'.  Assert DI.
0129                      clkit                   ;Clock the 93C46.
0130              m
0131              m                               ;93C46 device.
0132  0006  0565  m       bsf     serial,clock    ;Clock (CLK) = '1'.
0133              m
0134  0007  0000  m       nop                     ;Adjust the number of nop instructions
```


© 1993 Microchip Technology Incorporated

```
0135                m                                   ;between the assertion and de-assertion of
0136                m                                   ;CLK in proportion to the PIC16/17 operating
0137                m                                   ;frequency.  Refer to the 93C46 data for the
0138                m                                   ;minimum CLK period.
0139                m
0140   0008  0465           bcf     serial,clock        ;Clock (CLK) = '0'.
0141   0009  02F4           decfsz  cnt                 ;Repeat until cnt = 0.
0142   000A  0A02           goto    d_o_8               ;Cnt still > 0.
0143   000B  0360           rlf     indir               ;Restore register to its original condition.
0144   000C  0425           bcf     serial,din          ;make din low before return
0145   000D  0800           retlw   no_err              ;Exit with good status.
0146
0147                        ;*********************************************************************
0148                        ;*                              DIN8                                 *
0149                        ;*********************************************************************
0150                                                    ;Din8 will input 8 bits of data from the
0151                                                    ;93C46.  Before calling this routine, the FSR
0152                                                    ;must point to the register being used to
0153                                                    ;hold the incomming data.
0154                 din8
0155                        ;*********************************************************************
0156                                                    ;set up RA1 as a input before proceeding
0157   000E  0C02           movlw   b'00000010'         ;set up PORTA
0158   000F  0005           tris    serial              ;       /
0159                        ;*********************************************************************
0160   0010  0C08           movlw   0x08                ;Initialize loop counter.
0161   0011  0034           movwf   cnt                 ;
0162
0163                 d_i_8  clkit                       ;Clock a bit out of the 93C46.
0164                m
0165                m                                   ;93C46 device.
0166   0012  0565 m         bsf     serial,clock        ;Clock (CLK) = '1'.
0167                m
0168   0013  0000 m         nop                         ;Adjust the number of nop instructions
0169                m                                   ;between the assertion and de-assertion of
0170                m                                   ;CLK in proportion to the PIC16/17 operating
0171                m                                   ;frequency.  Refer to the 93C46 data for the
0172                m                                   ;minimum CLK period.
0173                m
0174   0014  0465           bcf     serial,clock        ;Clock (CLK) = '0'.
0175   0015  0360           rlf     indir               ;Make room for the incomming bit in the
0176                                                    ;destination register.
0177   0016  0400           bcf     indir,0             ;Assume that the incomming bit is a '0' and
0178                                                    ;clear the LSB of the destination register.
0179   0017  0625           btfsc   serial,dout         ;Test the incomming bit, if our assumption
0180                                                    ;was correct, skip the next instruction.
0181   0018  0500           bsf     indir,0             ;No, actual bit is a '1'.  Set the LSB of the
0182                                                    ;destination register.
0183   0019  02F4           decfsz  cnt                 ;Repeat until cnt = 0.
0184   001A  0A12           goto    d_i_8               ;Cnt still > 0
0185                        ;*********************************************************************
0186                                                    ;Restore RA1 back as output
0187   001B  0C00           movlw   0                   ;set RA1 as output
0188   001C  0005           tris    serial              ;       /
0189                        ;*********************************************************************
0190   001D  0800           retlw   no_err              ;Exit with good status.
0191
0192                        ;*********************************************************************
0193                        ;*                              RDYCHK                               *
0194                        ;*********************************************************************
0195                                                    ;Rdychk will read the 93C46 READY/BUSY status
0196                                                    ;and wait for RDY status within the alloted
0197                                                    ;number of processor cycles.  If RDY status
0198                                                    ;is not present after this set period, the
0199                                                    ;routine will return with an error status.
0200
0201                 rdychk
0202                        ;*********************************************************************
0203                                                    ;set up RA1 as a input before proceeding
0204   001E  0C02           movlw   b'00000010'         ;set up PORTA
```


© 1993 Microchip Technology Incorporated

```
0205  001F  0005          tris    serial          ;       /
0206                      ;*********************************************************************
0207  0020  0C04          movlw   tries           ;Initialize time-out counter.
0208  0021  0033          movwf   cnthi           ;
0209  0022  0074          clrf    cnt             ;
0210                      dsel                    ;De-select the 93C46.
0211              m
0212  0023  0405 m                 bcf     serial,cs ;Chip Select (CS) = '0' to de-select the
0213                                                  ;device.
0214
0215                ;       nop                     ;NOTE:  Check the 93C46 data sheet for
0216                                                  ;minimum CS low time.  Depending upon
0217                                                  ;processor frequency, a nop(s) may be
0218                                                  ;between the assertion and de-assertion of
0219                                                  ;Chip Select.
0220
0221                      sel                     ;Re-select the 93C46.
0222              m
0223  0024  0505          bsf     serial,cs  ;Chip Select (CS) = '1' to select the device.
0224  0025  0625  notrdy  btfsc   serial,dout       ;If DO is a '0', 93C46 has yet to completed
0225                                                  ;the last operation (still busy).
0226  0026  0A2E          goto    rdynoerr        ;skip to no error
0227  0027  02F4          decfsz  cnt             ;No, not yet ready.  Decrement the LSB of our
0228                                                  ;16 bit timer and check for expiration.
0229  0028  0A25          goto    notrdy          ;Still some time left.  Try again.
0230  0029  02F3          decfsz  cnthi           ;Least significant byte expired - decrement
0231                                                  ;and check for expiration of the MSB.
0232  002A  0A25          goto    notrdy          ;Still some time left.  Try again.
0233                      ;*********************************************************************
0234                                              ;Restore RA1 back as output
0235  002B  0C00          movlw   0               ;set RA1 as output
0236  002C  0005          tris    serial          ;       /
0237                      ;*********************************************************************
0238  002D  0801          retlw   error           ;RDY status was not present in the alloted
0239                                              ;time, return with error status.
0240              rdynoerr
0241                      ;*********************************************************************
0242                                              ;Restore RA1 back as output
0243  002E  0C00          movlw   0               ;set RA1 as output
0244  002F  0005          tris    serial          ;       /
0245                      ;*********************************************************************
0246  0030  0800          retlw   no_err
0247                      ;
0248                      ;
0249                      ;*********************************************************************
0250                      ;*                      SEE                                          *
0251                      ;*********************************************************************
0252
0253                                              ;See will control the entire operation of a
0254                                              ;93C46 device.  Prior to calling the routine,
0255                                              ;load a valid command/memory address into
0256                                              ;location cmd, and for WRITE or WRAL
0257                                              ;commands, load registers highb and lowb with
0258                                              ;16 bits of write data.  Upon exit, the W
0259                                              ;register will contain the completion status.
0260                                              ;Only 93C46 instructions which require a
0261                                              ;status check can return with an error as the
0262                                              ;completion status.  The values that denote
0263                                              ;the completion status are defined as
0264                                              ;variables 'error' and 'no_err' in the
0265                                              ;general assignments section.
0266
0267  0031  0C10  see     movlw   cmd                     ;Load W with the location of the cmd
0268                                              ;register.
0269  0032  0024          movwf   fsr             ;Transfer that information into the File
0270                                              ;Select Register.  The fsr now points to
0271                                              ;location cmd.
0272                      sel                     ;Select the 93C46.
0273              m
0274  0033  0505          bsf     serial,cs       ;Chip Select (CS) = '1' to select the device.
0275                      strtbt                  ;Send a start bit.
```


© 1993 Microchip Technology Incorporated

```
0276                    m
0277   0034  0525 m     bsf     serial,din      ;Start Bit = '1'.
0278                    clkit                   ;Clock it out.
0279                    m
0280                    m                       ;93C46 device.
0281   0035  0565 m     bsf     serial,clock    ;Clock (CLK) = '1'.
0282                    m
0283   0036  0000 m     nop                     ;Adjust the number of nop instructions
0284                    m                       ;between the assertion and de-assertion of
0285                    m                       ;CLK in proportion to the PIC16/17 operating
0286                    m                       ;frequency.  Refer to the 93C46 data for the
0287                    m                       ;minimum CLK period.
0288                    m
0289   0037  0465       bcf     serial,clock    ;Clock (CLK) = '0'.
0290   0038  0900       call    dout8           ;Transmit the 2 bit command and six bit
0291                                            ;address.
0292   0039  06D0       btfsc   cmd,6           ;Check for a WRITE or ERASE command.
0293   003A  0A43       goto    see2            ;Yes, parse the command further.
0294   003B  06F0       btfsc   cmd,7           ;Check for a READ command.
0295   003C  0A4D       goto    read_           ;Yes, process READ command.
0296   003D  06B0       btfsc   cmd,5           ;Check for a EWEN or ERAL command.
0297   003E  0A4A       goto    see3            ;Yes, parse the command further.
0298   003F  0690       btfsc   cmd,4           ;Check for a WRAL command.
0299   0040  0A52       goto    write_          ;Yes, process WRITE/WRAL command.
0300
0301                exit_dsel                   ;No further processing required; 93C46
0302                    m
0303   0041  0405 m     bcf     serial,cs       ;Chip Select (CS) = '0' to de-select the
0304                                            ;device.
0305                                            ;command completed.
0306   0042  0800       retlw   no_err          ;Return with good completion status.
0307
0308   0043  07F0 see2   btfss   cmd,7          ;Check for a ERASE command.
0309   0044  0A52       goto    write_          ;No, process WRITE command.
0310   0045  091E exit2_ call    rdychk          ;ERASE command requires a status check.
0311                    dsel                    ;De-select the 93C46.
0312                    m
0313   0046  0405 m     bcf     serial,cs       ;Chip Select (CS) = '0' to de-select the
0314                                            ;device.
0315   0047  01E2       addwf   pc              ;Compute completion status from results of
0316                                            ;status check.
0317   0048  0800       retlw   no_err          ;Return with good completion status.
0318   0049  0801       retlw   error           ;Return with bad completion status.
0319
0320   004A  0690 see3   btfsc   cmd,4          ;Check for a EWEN command.
0321   004B  0A41       goto    exit_           ;Yes, no further processing required, exit
0322                                            ;now.
0323   004C  0A45       goto    exit2_          ;No, ERAL command which requires a status
0324                                            ;check.
0325
0326   004D  02A4 read_  incf    fsr             ;Increment the File Select Register to point
0327                                            ;to the register receiving the upper byte of
0328                                            ;the incomming 93C46 data word.
0329   004E  090E       call    din8            ;Input the upper byte.
0330   004F  02A4       incf    fsr             ;Increment the File Select Register to point
0331                                            ;to the register receiving the lower byte.
0332   0050  090E       call    din8            ;Input 8 more bits.
0333   0051  0A41       goto    exit_           ;No further processing required, exit now.
0334
0335   0052  02A4 write_ incf    fsr             ;Increment the File Select Register to point
0336                                            ;to the upper byte of the 16 bit 93C46 data
0337                                            ;word to be transmitted.
0338   0053  0900       call    dout8           ;Output that byte.
0339   0054  02A4       incf    fsr             ;Increment the File Select Register to point
0340                                            ;to the lower byte.
0341   0055  0900       call    dout8           ;Output the lower byte of the 16 bit 93C46
0342                                            ;data word.
0343   0056  0A45       goto    exit2_          ;Exit with a status check.
0344
0345                    ;*****************************************************************
```


© 1993 Microchip Technology Incorporated

```
0346                            ;*                      Test Program          *
0347                            ;*****************************************************************
0348                            main                    ;We've include a sample program to exercise
0349                                                    ;the PIC16/17 to 93C46 interface using a simple
0350                                                    ;erase, write and varify routine.
0351
0352   0057  0065               clrf    serial         ;Clear the port tied to the 93C46 device.
0353   0058  0CF4               movlw   b'11110100'    ;Intialize the data direction register for
0354   0059  0005               tris    serial         ;that port.
0355
0356   005A  0C30               movlw   ewen           ;Load W with the Erase/Write Enable command.
0357   005B  0030               movwf   cmd            ;Transfer W into cmd register.
0358   005C  0931               call    see            ;Enable the 93C46 device.
0359
0360   005D  0C20               movlw   eral           ;Load W with the Erase All command.
0361   005E  0030               movwf   cmd            ;Transfer W into cmd register.
0362   005F  0931               call    see            ;Erase the 93C46.
0363   0060  0F01               xorlw   error          ;Check completion status.
0364   0061  0643               btfsc   status, zflag  ;Test for error condition.
0365   0062  0A82               goto    errloop        ;Yes, bad completion status, error-out.
0366                                                   ;Write loop:
0368          001F  loopcnt equ      0x1F             ;Define an unused location for our test
0369                                                          ;program loop counter.
0370          00AA  tstptrn equ      0xAA             ;Define the test pattern to be written.
0371
0372   0063  0C40               movlw   .64            ;Initialize that counter.
0373   0064  003F               movwf   loopcnt        ;
0374   0065  0C40               movlw   write          ;Load W with the Write command.
0375   0066  0030               movwf   cmd            ;Transfer W into cmd register.
0376   0067  0CAA               movlw   tstptrn        ;Intialize the 93C46 data registers with
0377                                                   ;write data.
0378   0068  0031               movwf   highb          ;
0379   0069  0032               movwf   lowb           ;
0380   006A  0931  test1        call    see            ;Write data word into 93C46 device.
0381   006B  0F01               xorlw   error          ;Check completion status.
0382   006C  0643               btfsc   status,zflag   ;Test for error condition.
0383   006D  0A82               goto    errloop        ;Yes, bad completion status, error-out.
0384   006E  02B0               incf    cmd            ;No, increment the 6 bit memory address
0385                                                   ;field.
0386   006F  02FF               decfsz  loopcnt        ;Have we written all 64 locations?
0387   0070  0A6A               goto    test1          ;No, write another location.
0388
0389                                                   ;Read loop:
0390
0391   0071  0C40               movlw   .64            ;Initialize loop counter.
0392   0072  003F               movwf   loopcnt        ;
0393   0073  0C80               movlw   read           ;Load W with the Read command.
0394   0074  0030               movwf   cmd            ;Transfer W into cmd register.
0395   0075  0931  test2        call    see            ;Read addressed data word from 93C46 device.
0396   0076  0CAA               movlw   tstptrn        ;Load W with the pattern written.
0397   0077  0091               subwf   highb,0        ;Verify the data read against what was
0398                                                   ;written.
0399   0078  0743               btfss   status,zflag   ;Same?
0400   0079  0A82               goto    errloop        ;No, error-out.
0401   007A  0CAA               movlw   tstptrn        ;Repeat with the lower byte read.
0402   007B  0092               subwf   lowb,0         ;
0403   007C  0743               btfss   status,zflag   ;Same?
0404   007D  0A82               goto    errloop        ;No, error-out.
0405   007E  02B0               incf    cmd            ;Yes, both byte correct, increment the 6 bit
0406                                                   ;memory address field.
0407   007F  02FF               decfsz  loopcnt        ;Have we read all 64 locations?
0408   0080  0A75               goto    test2          ;No, read another location.
0409
0410   0081  0A81  allok        goto    allok          ;Home safe!
0411
0412   0082  0A82  errloop      goto    errloop        ;Bad news!
0413
0414          0000               END                    ;Thats all folks!
```

© 1993 Microchip Technology Incorporated

NOTES:

© 1993 Microchip Technology Incorporated

Logic Powered Serial EEPROMs

Embedded applications increasingly want more integration and power, in less space for less cost. Using low power Serial EEPROMs (SEE) for application firmware, lookup tables, and microcode coupled with small footprints makes for permanent storage at respectable savings. One additional method of saving on the power budget is selectively powering off components when not needed, a basic for embedded power management. The low-power SEEs offered by Microchip Technology Inc., offer an additional benefit, powering the SEE from a microcontroller port. This allows the host controller to not only manipulate the Serial EEPROM Reads and Write, but also the periods when it is powered off or on. Satellite communications use this technique to save power and total dose accumulation. We call this technique POWER PORT™. The microcontroller port must have sufficient Ioh (source current) to sustain the voltage and current for all memory functions, READ, ERASE, and WRITE. Obviously, not all memory or peripheral devices could be powered thusly, but Microchip's SEE devices will function in this environment.

The microcontroller, using its internal software and hardware decision functions, determines when it needs to communicate with the memory device, then acts accordingly. Any standard wake-up sequence will accomplish this task. The wake-up code needs only power up the memory and wait for the power to become stable before doing a read or write by driving the POWER PORT high. Then all serial communication executes normally. The SEEs are powered off for additional power savings and the data or code is utilized from RAM. Obviously, the port output must be allowed to settle, but normal operation of the output structures would guarantee that this would be met. The I/O port Tpd for the Microchip PIC16C5X, is specified at 40ns maximum.

The 24LCXX and 93LCXX CMOS SEE series parts from Microchip were designed to achieve low current consumption across all ranges of operation.

The four primary Icc parameters for these products are:

Parameter	Conditions
Icc STANDBY	Not in an active operation while Vcc is supplied.
Icc READ	The part is in a READ operation.
Icc PEAK WRITE	The BYTE / PAGE WRITE and ERASE operations have self timed cycles of 10 ms. A typical of 4 ms is the actual time of the operation. This is the amount of time when the Icc requires the most current (PEAK WRITE). The part is drawing STANDBY Icc during the remaining 6ms of the cycle.
Icc AVG WRITE	The avg of the PEAK WRITE Icc and STANDBY Icc during the self-timed 10ms write cycle.

The attached characteristic curves (Figures A and B) indicate that Icc PEAK WRITE current consumes the most current. The worst case condition is at 6.0V and –40°C. The 24LCXX series parts draw a typical 3.2 mA and the 93LCXX series parts draw a typical of 2.0 mA. These low Icc characteristics offer a unique current saving benefit for battery applications. Figure C and D illustrate the sink and source current capabilities of the PIC16C5X family of microcontrollers. It is clear from these characterization curves that the microcontroller can deliver sufficient current across all temperature ranges to power a SEE using the POWER PORT technique.

Figure E shows the connection scheme for the Microchip PIC16C54. It should be noted that not all versions of competitive microcontrollers are capable of powering a device in this manner and the specific data sheets for the microcontroller being considered must be consulted for maximum source current. The microcontroller port must be capable of sourcing sufficient current for the duration of the write cycle or 10ms, worse case. The peak write requirement for the 24LCXX product family is 3.2 mA at 5.5 Vdc (–40°C).

Listing A demonstrates the appropriate code sequences when using the PIC16C54 microcontroller. The sequences included are power control, start bit, stop bit, send and receive bit, Tx and Rx, and a general addressing routine.

© 1993 Microchip Technology Incorporated

FIGURE A - TYPICAL Icc FOR 24LCXX

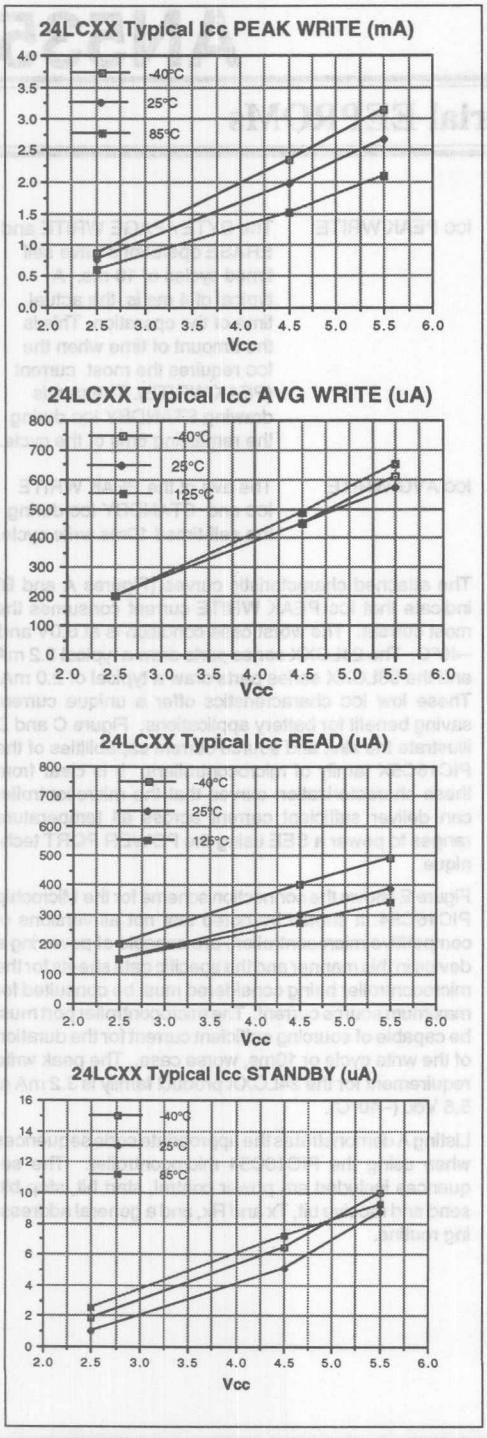
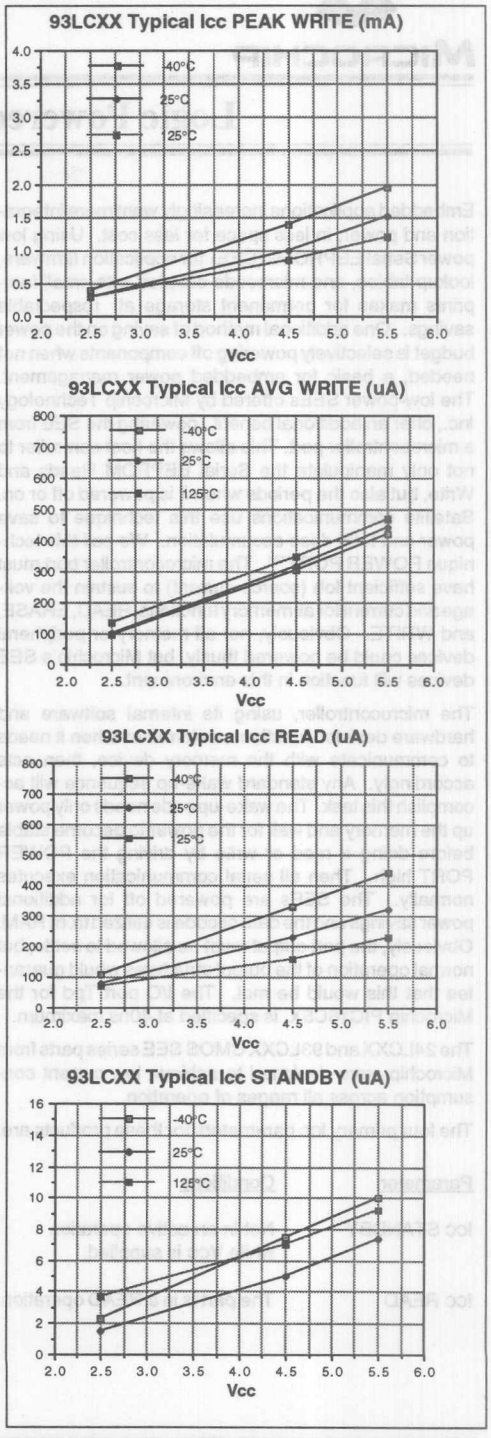


FIGURE B - TYPICAL Icc FOR 93CXX



© 1993 Microchip Technology Incorporated

FIGURE C - PIC16C5X I_{OL} AT 5V

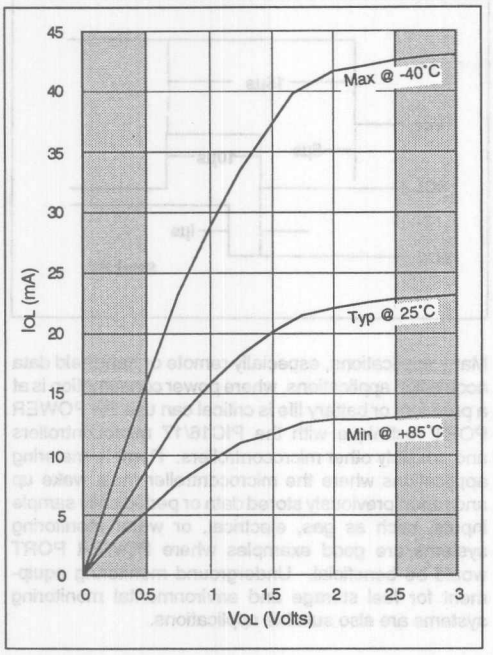


FIGURE D - PIC16C5X I_{OH} AT 5V

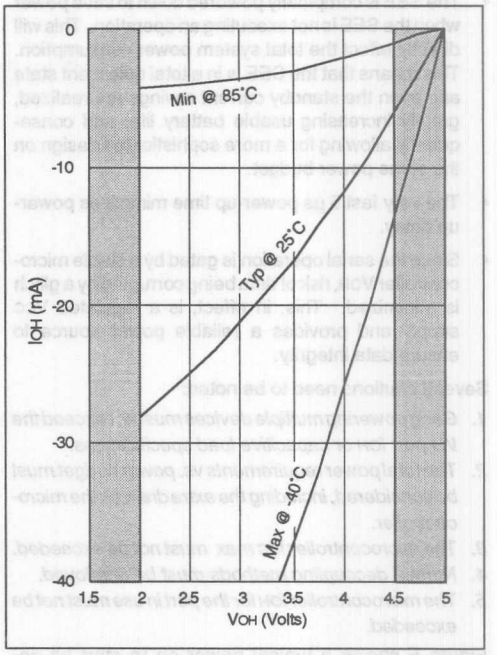
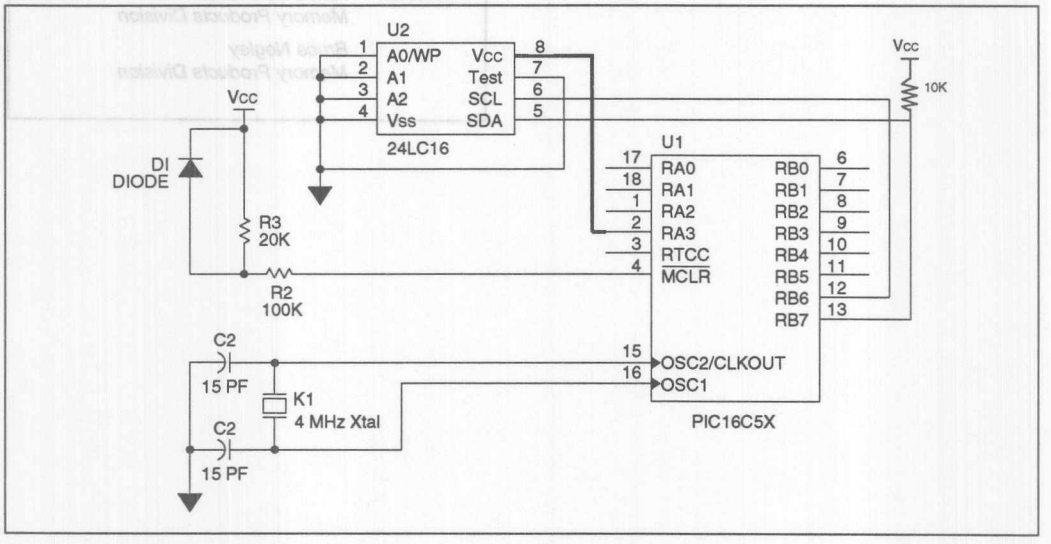


FIGURE E - 24LC16/PIC16C5X INTERFACE SCHEMATIC



7

The primary benefits of this application are:

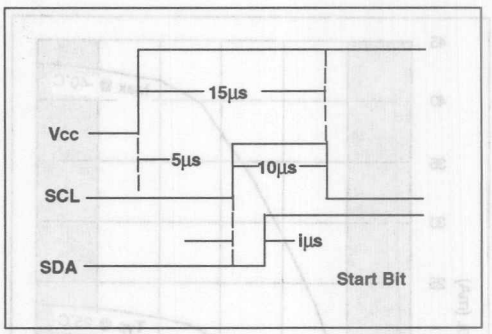
- The SEE is completely powered down to save power when the SEE is not executing an operation. This will directly effect the total system power consumption. This means that the SEE is in a total quiescent state and even the standby current savings are realized, greatly increasing usable battery life, and consequently allowing for a more sophisticated design on the same power budget.
- The very fast 5 µs power-up time minimizes power-up delay.
- Since the serial operation is gated by a stable microcontroller V_{OH}, risk of data being corrupted by a glitch is minimized. This, in effect, is a regulated V_{CC} supply and provides a reliable power source to ensure data integrity.

Several cautions need to be noted:

1. *Gang powering multiple devices must not exceed the I/O port I_{OH} or capacitive load specifications.*
2. *The total power requirements vs. power budget must be considered, including the extra drain on the microcontroller.*
3. *The microcontroller I_{CC} max must not be exceeded.*
4. *Normal decoupling methods must be employed.*
5. *The microcontroller I_{OH} for the port in use must not be exceeded.*

Figure F shows a typical power on to start bit sequence. Notice that the device is available to receive a clock at 5 µs after V_{CC} has become stable.

FIGURE F



Many applications, especially remote or handheld data acquisition applications, where power consumption is at a premium or battery life is critical can use the POWER PORT technique with the PIC16/17 microcontrollers and possibly other microcontrollers. Remote metering applications where the microcontroller must wake up and report previously stored data or periodically sample inputs, such as gas, electrical, or water monitoring systems are good examples where POWER PORT would be beneficial. Underground monitoring equipment for fuel storage and environmental monitoring systems are also suitable applications.

Authors: *Richard J. Fisher*
Memory Products Division
Bruce Negley
Memory Products Division

© 1993 Microchip Technology Incorporated

LISTING A

```
LIST P=16C54
;
;
;    Sample test program to power up serial EEPROM
;    using PIC16/17 port A, then write one byte and read same byte, then repeat forever.
;
;*****************************************************************************************
port_a  equ     5h      ; port 5 used for device
                        ; address select
port_b  equ     6h      ; port 6 used for data and
                        ; clock lines
eeprom  equ     0ah     ; bit buffer
addr    equ     0ch     ; address register
datai   equ     0dh     ; stored data input reg.
datao   equ     0eh     ; stored data output reg.
slave   equ     0fh     ; device address
                        ; (1010xxx0)
txbuf   equ     10h     ; tx buffer
count   equ     11h     ; bit counter
bcount  equ     12h     ; byte counter
rxbuf   equ     13h     ; receive buffer
loops   equ     15h     ; delay loop counter
loops2  equ     16h     ; delay loop counter 2
;
;       Bit Assignments
;
di      equ     7               ; eeprom input
do      equ     6               ; eeprom output
sdata   equ     7               ; data line (port_b)
sclk    equ     6               ; clock line (port_b)
vcc     equ     3               ; vcc for dut (port_a)
;
        org     01ffh
begin   goto    PWRUP
        org     000h
        goto    PWRUP
;
;*****************************************************************************************
;       DELAY ROUTINE
;       this routine takes the value in loops and loops that many times.  Every
;       increase in 'loops' yields approx 1 more millisecond.
;       i.e., if 'loops' is 10 then the wait period is approx 10 milliseconds.
;
;-----------------------------------------------------------------------------------------
WAIT
;

top2    movlw   .110
        movwf   loops2
top     nop             ; sit and wait
        nop
        nop
        nop
        nop
        nop
        decfsz  loops2  ; inner loop done?
        goto    top     ; no, go again
        decfsz  loops   ; outer loop done?
        goto    top2    ; no, go again
        retlw   0       ; yes, return from sub
;
;*****************************************************************************************
```

7

© 1993 Microchip Technology Incorporated

```
;       Start Bit Subroutine
;       this routine generates a start bit
;-------------------------------------------------------------------------------
;
BSTART
        movlw   b'00111111'
        tris    port_b          ; port b for output
        bsf     port_b,sdata    ; set clock high
        nop
        nop
        bsf     port_b,sclk     ; set clock high
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sdata    ; data line goes low during high clock for start bit
        nop
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; start clock train
        nop
        nop
        nop
        retlw   0
         ;
         ;    End of Subroutine
;*******************************************************************************
;
;       Stop Bit Subroutine
;       this routine generates a stop bit
;-------------------------------------------------------------------------------
BSTOP
        movlw   b'00111111'     ;
        tris    port_b          ; set data/clock lines as outputs
        bcf     port_b,sdata    ; make sure data line is low
        nop
        nop
        nop
        nop
        nop
        nop
        bsf     port_b,sclk     ; set clock high
        nop
        nop
        nop
        nop
        nop
        nop
        bsf     port_b,sdata    ; data goes high while clock high
                                ; for stop bit
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; set clock low again
        nop
        nop
        nop
        retlw   0
;
;           End of Subroutine
;*******************************************************************************
```


© 1993 Microchip Technology Incorporated

```
;       Serial data send 1 bit from PIC16/17 to dut
;-------------------------------------------------------------------------------
BITOUT
        movlw   b'00111111'     ; set data,clock as outputs
        tris    port_b
        btfss   eeprom,do
        goto    BIT0
        bsf     port_b,sdata    ; output bit 0
        goto    CLK1            ; data line clocked low by device
;
BIT0
        bcf     port_b,sdata    ; output bit 0
CLK1
        nop
        nop
        bsf     port_b,sclk     ; set clock line high
BIT2
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; return clock line low
        retlw   0
;
;       End of Subroutine
;
;*******************************************************************************
;       Bit in routine
;       this routine gets a bit of data from the part
;        into the 'eeprom' register, bit 'di'
;-------------------------------------------------------------------------------
BITIN
        movlw   b'10111111'     ; make sdata an input line
        tris    port_b
        bcf     eeprom,di       ; assume input bit low
        bsf     port_b,sclk     ; set clock line high
        nop                     ; just sit here a sec
        nop
        nop
        nop
        nop
        nop
        nop
        nop                     ;
        btfsc   port_b,sdata    ; read data line
        bsf     eeprom,di       ; set input bit if needed
        bcf     port_b,sclk     ; set clock line low
        retlw   0               ; hit the road
;
;*******************************************************************************
;
;       Transmit Data Subroutine
;-------------------------------------------------------------------------------
TX
        movlw   .8
        movwf   count           ; set the #bits to 8
        ;
TXLP
        bcf     eeprom,do
        btfsc   txbuf,7
        bsf     eeprom,do       ; otherwise data bit =1
        call    BITOUT          ; serial data out
        rlf     txbuf           ; rotate txbuf left
        decfsz  count           ; 8 bits done?
        goto    TXLP    ; no - go again
        call    BITIN   ; read ack bit
        ;
        retlw   0
;       End of Subroutine
;*******************************************************************************
```


© 1993 Microchip Technology Incorporated

```
;       Receive data Routine
;       this routine gets a byte of data from the part into 'rxbuf'
;-------------------------------------------------------------------------------
RX
        movlw   .8              ; set # bits to 8
        movwf   count
        clrf    rxbuf           ; clear receive buffer
RXLP    rlf     rxbuf           ; rotate buffer left 1 bit
        bcf     rxbuf,0         ; assume bit is zero
        call    BITIN           ; read a bit
        btfsc   eeprom,di       ; input bit high?
         bsf    rxbuf,0         ; yes, set buffer bit high
        decfsz  count           ; 8 bits done?
        goto    RXLP            ; no, do another
        bcf     eeprom,do       ; set ack bit = 0
        call    BITOUT          ; to finish transmission
        retlw   0
;
;*******************************************************************************
;       Power up routine
;       this routine blinks the lights
;-------------------------------------------------------------------------------
PWRUP
        movlw   b'00000001'
        tris    port_a          ; set RA0 as input, rest output
        bsf     port_a,vcc      ; turn on power to dut
        nop                     ; wait for dut to power up
        nop
        nop
        nop
        nop
;
;*******************************************************************************
;       Byte Write Routine
;       this writes the data in "55h" to the first byte
;       in the serial EEPROM.
;-------------------------------------------------------------------------------
;
WRBYTE
                                ;
        movlw   b'10100000'     ; set slave address and write mode
        movwf   slave
        movlw   b'01010101'     ; set data to 55h
        movwf   datao
        ;
        clrf    addr            ; set address to 00h
        ;
        call    BSTART          ; generate start bit
        movf    slave,w         ; get slave address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        movf    addr,w          ; get word address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        movf    datao,w         ; move data
        movwf   txbuf           ; to tranmit buffer
        call    TX              ; and transmit it
        call    BSTOP           ; generate stop bit
                                ;
        movlw                   .10
        movwf   loops   ; set delay time to give
        call    WAIT        ; 10 ms wait after every byte
                        ;
                        ; now drop through and do the read
                        ;
;*******************************************************************************
;       READ (read routine)
;       this routine reads the first address
;       of the dut
```


© 1993 Microchip Technology Incorporated

```
;-------------------------------------------------------------------------
READ
                                ;
        movlw   b'10100000'     ; set slave address and write mode
        movwf   slave
                                ;
        clrf    addr            ; set address to 00h
                                ;
        call    BSTART          ; generate start bit
        nop
        nop
        movf    slave,w         ; get slave address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        movf    addr,w          ; get word address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        nop
        nop
        call    BSTART          ; generate start bit
        nop
        nop
        movlw   b'10100001'     ; get slave address and read mode
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and transmit it
        nop
        call    RX              ; get 8 bits of data
        bsf     eeprom,do
        call    BITOUT          ; send high ack bit and then a
        call    BSTOP           ; stop bit to end transmission from dut
        nop                     ;
        nop
        nop
        nop
        nop
        bcf     port_a,vcc      ; turn power to dut off
        movlw   .100
        movwf   loops
        call    WAIT            ; wait awhile
        goto    PWRUP           ; go do the whole thing over again
;
END
```

7

© 1993 Microchip Technology Incorporated

NOTES:

© 1993 Microchip Technology Incorporated

SECTION 8
SERIAL EEPROMS
TUTORIALS AND APPLICATION NOTES

Basic Serial EEPROM Operation - AN536 ..... 8- 1
Everything a System Engineer Needs to Know About
Serial EEPROM Endurance - AN537 ..... 8- 15
Using the Microchip Endurance Predictive Software - AN562 ..... 8- 23
Interfacing 24LCXXB Serial EEPROMs to the PIC16C54 - AN567 ..... 8- 27
Using the 24XX65 and 24XX32 with Stand-alone PIC16C54 Code - AN558 ..... 8- 53
24C01A Compatibility Issue and Its Mobility for Memory Upgrade - AN517 ..... 8- 93
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559 ..... 8- 95
Using the 93LC56 and 93LC66 - AN560 ..... 8- 99
1.8 Volt Technology - Benefits - AN550 8-119
Serial EEPROM Solutions vs. Parallel Solutions - AN551 8-121
Questions and Answers Concerning Serial EEPROMs - AN572 8-125

© 1994 Microchip Technology Inc.

8

© 1994 Microchip Technology

MICROCHIP

AN536

Basic Serial EEPROM Operation

BASIC SERIAL EEPROM OPERATION

Looking for the optimum non-volatile memory product for your system that requires a small footprint, byte level flexibility, low power, and is highly cost effective? Serial EEPROM technology is one of the non-volatile memory technologies that has emerged as a leading embedded control solution. Unfortunately, most system designers are not aware of the serial EEPROM benefits. Also, the supporting documentation in databooks is most often not adequate due to incomplete or ambiguous information. As a result, the system designer often selects a non-volatile solution that does not meet his requirements, or, the designer must face a more complicated design-in with a serial EEPROM.

This article addresses two issues that exist today for designers considering serial EEPROM products:

First, to provide awareness of the application benefits.

Secondly, to provide a primer on the operating principles and instructions. These items are often buried in databook text or not adequately addressed. Also included are common default conditions to significantly reduce the system designer's learning curve.

CONTENTS

Serial EEPROM Applications

Overview of the Primary Protocol Benefits

3-Wire Bus Operation Primer

2-Wire Bus Operation Primer

Microchip 2-Wire Default Conditions

Timing Diagram Attachments

SERIAL EEPROM APPLICATIONS

Serial EEPROMS are ideal non-volatile cost effective memory solutions in applications that require:

- Small footprint and board space as in cellular phone applications
- BYTE level ERASE, WRITE, and READ of data as in a TV tuner
- Low voltage and current for handheld battery applications as in a keyless entry transmitter
- Multiple non-volatile functions in the same application such as a VCR
- Low availability of microcontroller I/O lines

The common applications for Serial EEPROMS are shown below:

Market	Common Applications
Consumer	TV tuners, VCRs, CD players, cameras, radios, and remote controls
Automotive	Airbags, anti-lock brakes, odometers, radios, and keyless entry
Office Automation	Printers, copiers, PCs, and portable PCs
Telecom	Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers
Industrial	Bar code readers, point-of-sale terminals, smart cards, lock boxes, garage door openers, and test measurement equipment

The typical functions that serial EEPROMs are utilized for are:

- Memory storage of channel selectors or analog controls (volume, tone, etc.) in consumer electronics products
- Power down storage and retrieval of events such as fault detection or error diagnostics in automotive products
- Electronic real time event or maintenance logs such as page counting in office automation products. Also, configuration or DIP switch storage in office automation products
- Last number redial storage and speed dial number storage in telecom products
- User in-circuit reprogrammable look up tables such as bar code readers, point-of-sale terminals, environmental controls and other industrial products

Other application examples include:

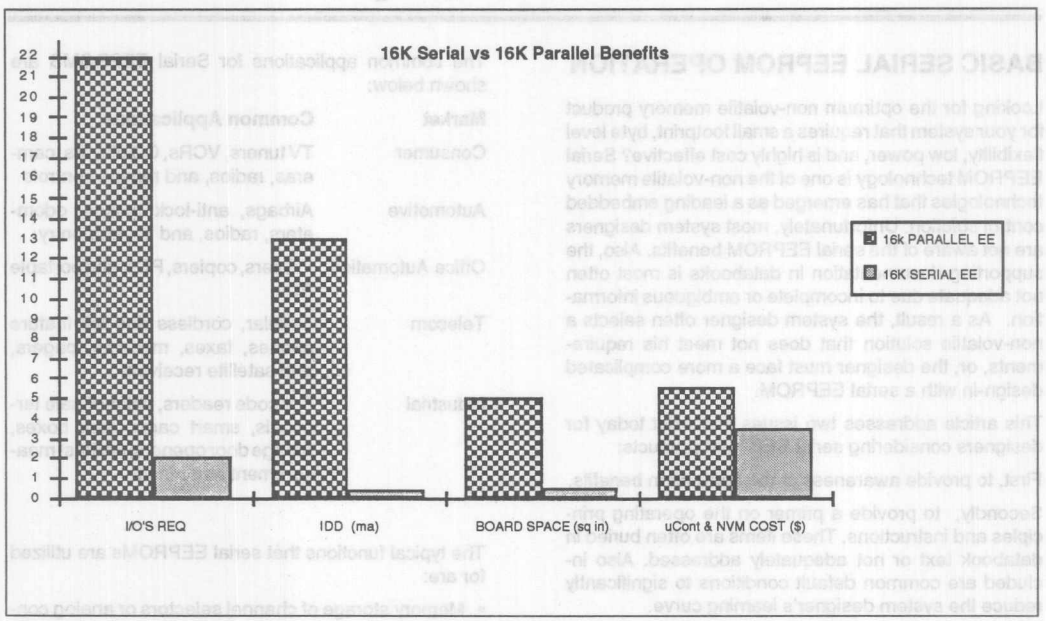
- Data storage from a learn function as in a remote control transmitter
- ID number storage for security or remote access for electronic keys and entry databases
- Reprogrammable calibration data for test equipment or analog interface products

8

© 1993 Microchip Technology Inc.

Basic Serial EEPROM Operation

As a result of density and architectural evolution, Serial EEPROMs offer significant benefits in some applications that previously could only utilize Parallel EEPROM products. The diagram below illustrates the footprint and board space differences.



The Serial EEPROM requires only 10% of the board space that a Parallel EEPROM requires. Also, the Serial EEPROM requires fewer I/O lines from the microcontroller which significantly reduces the overall system cost and board space.

A very fast READ speed is the only significant limitation of a Serial EEPROM for a decision between a serial and a Parallel EEPROM. It is very interesting to note that the Serial EEPROM READ speed is restricted more by the protocol than the process technology. The 2-wire I²C (Inter-Integrated Circuit) products must add large internal delays to slow down the part to meet the 100KHz protocol requirements, which will be reviewed later. Characterization of 3-wire bus Serial EEPROMs have indicated clock frequencies in excess of 6MHz.

OVERVIEW OF THE PRIMARY PROTOCOL BENEFITS

After a designer decides to use a serial EEPROM solution, the next step is to select one of the two primary serial EEPROM protocols. Unfortunately, most system designers select the type of serial EEPROM (2- or 3-wire) that they are most familiar with, regardless of the benefits associated with each type.

© 1993 Microchip Technology Inc.

The benefits of each protocol are shown below:

3-Wire Bus Serial EEPROMS	2-Wire Bus Serial EEPROMS
Single VDD supply of <2V to 5.5V	Single VDD supply of <2V to 5.5V
Very low current consumption	Very low current consumption
Reduced overall component cost	Reduced overall component cost
Four pins (other than VCC & GND) are required or operation	Two pins (other than VCC & GND) are required for operation
x16 bit and x8 bit data widths	x8 data bit width
Software WRITE Protection	Hardware WRITE Protection
Edge triggered clocks and signals	Level triggered clocks and signals and input glitch filters for high noise immunity
2MHz+ operation	I²C standard 100KHz and 400KHz protocols with a 1MHz option
Ready/Busy data polling	Page WRITE capability to 16 bytes
Security options available	Software and hardware compatible from 2K to 16K densities
Less complex protocol	

A 2-wire product is utilized in applications that require an I²C bus, noise immunity, limited microcontroller I/O pin availability, or a WRITE buffer for multiple bytes to be stored with one instruction. A 3-wire product is utilized in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or x16 data width applications.

The next two sections describe the basic operation and Microchip's default conditions for the 3-wire and 2-wire Serial EEPROMs to allow the system designer to utilize the benefits of Serial EEPROMs.

3-WIRE BUS OPERATION PRIMER

Many serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. Serial EEPROMs are not conventional memories due to the Serial communication protocols involved. This section is a PRIMER for the data sheet to familiarize the system designer with the basic principles of the 3-wire bus operation.

Basic Principles

Common device nomenclature is 93XXXX.

The 93XX06 is a 256 bit product.

The 93XX46 is a 1K bit product.

The 93XX56 is a 2K bit product.

The 93XX66 is a 4K bit product.

Four pins are required:

CS (Chip Select)	DI (data in)
CLK (Clock)	DO (data out)

All 93XXXX parts are hardware compatible for these four pins. However, there may be compatibility issues for the other pins.

Even though there is hardware compatibility on the four pins, there can be differences from a software standpoint. Subtle differences between each manufacturer's products, referred to as default conditions, can prevent plug compatibility. These issues are addressed later in the attached 3-Wire Timing Diagram. There is no industry standardized upgrade path for density migration. Please review density upgrades for Microchip's products on a case-by-case basis.

Data is available in x8 or x16 organizations. This selection is determined either by the ORG pin or by purchasing a standard x16 organization.

Units will power-up in a EWDS (ERASE/WRITE Disable State). All ERASE and WRITE functions are disabled until the EWEN (ERASE/WRITE Enable) instruction is performed. This is to prevent accidental data corruption.

An Auto-ERASE (logical "1") cycle is performed during each WRITE Cycle.

The 7 instructions are shown in the attached instruction set table. These instructions are for Microchip's 93LCXX family products.

After an instruction is loaded, the CLK and DI pins are in a DON'T CARE state until the next START bit.

8

© 1993 Microchip Technology Inc.

The following is required for each instruction set (all input bits are triggered by the positive clock edges):

Start Bit	The first Data-in high signal clocked in after CS is high.
Opcode	Two Bits to identify the instruction
Address	Refer to the Instruction Set table for the number of bits required.
Data	Separate data-in and data-out pins. However, these two pins may be tied together for true 3-wire operation. Please refer to the attached 3-wire Bus READ timing diagram example.

READ, WRITE, and ERASE

The attached 93LC66 timing diagrams illustrate the key concepts and timing parameters for each of these operations. Please refer to the instruction set tables and the AC parameters in the databook for supplemental information.

ERASE ALL (ERAL)

An ERASE ALL (ERAL) operation is identified by a "00" opcode. The ERAL instruction requires the next two bits to be clocked in as "10" in the address block of the instruction set. All bits in the array will be set to a logic "1" state by one command in typically less than 10ms.

WRITE ALL (WRAL)

A WRITE ALL (WRAL) operation is also identified by a "00" opcode. The WRAL requires the next two bits to be clocked in as "01" in the address block of the instruction set. The data-in block will contain the data for a SINGLE BYTE which is to be repeated throughout the entire array. For example, if a 4F5A is loaded in the 16 data-in bits of the instruction set, a 4F5A will be written into every word in the array.

EWEN and EWDS

As stated before, all units will power up in to an ERASE/WRITE DISABLE (EWDS) state to prevent data corruption. All future ERASE/WRITE operations must execute an ERASE/WRITE ENABLE (EWEN) opcode until the next power down is detected or until other EWDS opcodes are executed. Please refer to the instruction set table.

© 1993 Microchip Technology Inc.

INSTRUCTION SET FOR 93LC46: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A5 A4 A3 A2 A1 A0	—	D15 - D0	25
EWEN	1	00	1 1 X X X X	—	High-Z	9
ERASE	1	11	A5 A4 A3 A2 A1 A0	—	(RDY/$\overline{BSY}$)	9
ERAL	1	00	1 0 X X X X	—	(RDY/$\overline{BSY}$)	9
WRITE	1	01	A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/$\overline{BSY}$)	25
WRAL	1	00	0 1 X X X X	D15 - D0	(RDY/$\overline{BSY}$)	25
EWDS	1	00	0 0 X X X X	—	High-Z	9

INSTRUCTION SET FOR 93LC46: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	18
EWEN	1	00	1 1 X X X X X	—	High-Z	10
ERASE	1	11	A6 A5 A4 A3 A2 A1 A0	—	(RDY/$\overline{BSY}$)	10
ERAL	1	00	1 0 X X X X X	—	(RDY/$\overline{BSY}$)	10
WRITE	1	01	A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/$\overline{BSY}$)	18
WRAL	1	00	0 1 X X X X X	D7 - D0	(RDY/$\overline{BSY}$)	18
EWDS	1	00	0 0 X X X X X	—	High-Z	10

INSTRUCTION SET FOR 93LC56: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A6 A5 A4 A3 A2 A1 A0	—	D15 - D0	27
EWEN	1	00	1 1 X X X X X X	—	High-Z	11
ERASE	1	11	X A6 A5 A4 A3 A2 A1 A0	—	(RDY/$\overline{BSY}$)	11
ERAL	1	00	1 0 X X X X X X	—	(RDY/$\overline{BSY}$)	11
WRITE	1	01	X A6 A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/$\overline{BSY}$)	27
WRAL	1	00	0 1 X X X X X X	D15 - D0	(RDY/$\overline{BSY}$)	27
EWDS	1	00	0 0 X X X X X X	—	High-Z	11

INSTRUCTION SET FOR 93LC56: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A7 A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	20
EWEN	1	00	1 1 X X X X X X X	—	High-Z	12
ERASE	1	11	X A7 A6 A5 A4 A3 A2 A1 A0	—	(RDY/$\overline{BSY}$)	12
ERAL	1	00	1 0 X X X X X X X	—	(RDY/$\overline{BSY}$)	12
WRITE	1	01	X A7 A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/$\overline{BSY}$)	20
WRAL	1	00	0 1 X X X X X X X	D7 - D0	(RDY/$\overline{BSY}$)	20
EWDS	1	00	0 0 X X X X X X X	—	High-Z	12

INSTRUCTION SET FOR 93LC66: ORG = 1 (x 16 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A7 - A0	—	D15 - D0	27
EWEN	1	00	11XXXXXX	—	High-Z	11
ERASE	1	11	A7 - A0	—	(RDY/$\overline{BSY}$)	11
ERAL	1	00	10XXXXXX	—	(RDY/$\overline{BSY}$)	11
WRITE	1	01	A7 - A0	D15 - D0	(RDY/$\overline{BSY}$)	27
WRAL	1	00	01XXXXXX	D15 - D0	(RDY/$\overline{BSY}$)	27
EWDS	1	00	00XXXXXX	—	High-Z	11

INSTRUCTION SET FOR 93LC66: ORG = 0 (x 8 organization)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A8 - A0	—	D7 - D0	20
EWEN	1	00	11XXXXXXX	—	High-Z	12
ERASE	1	11	A8 - A0	—	(RDY/$\overline{BSY}$)	12
ERAL	1	00	10XXXXXXX	—	(RDY/$\overline{BSY}$)	12
WRITE	1	01	A8 - A0	D7 - D0	(RDY/$\overline{BSY}$)	20
WRAL	1	00	01XXXXXXX	D7 - D0	(RDY/$\overline{BSY}$)	20
EWDS	1	00	00XXXXXXX	—	High-Z	12

8

© 1993 Microchip Technology Inc.

2-WIRE BUS OPERATION PRIMER

As indicated in the 3-wire bus section, many serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are, unfortunately, either vaguely embedded in the data sheet text or not included. This section is a PRIMER for the data sheet to familiarize the system designer with the basic 2-wire serial EEPROM operation principles.

Basic Principles

The common device nomenclature is 24XXXX and 85XXXX.

Only the SCL and SDA pins are essential for bus operation. The other pins are supplementary:

SCL (Serial clock)

SDA (Serial Data)

WP (Active High WRITE Protection)

A0, V A1, and A2 (Chip or block select)

SDA's open-drain requires a pull-up resistor to VDD.

The data is organized as x8.

Signals are level triggered, not edge triggered. Also, there are filters on the inputs that will filter noise glitches <100ns wide.

An Auto-ERASE (logical "1") cycle is performed during each WRITE cycle.

The I²C protocol utilizes master/slave bi-directional communication. A device that sends data onto the bus is defined as the transmitter, and a device that is receiving data is the receiver. Both the master and the slave can operate as the transmitter or receiver. The bus must be controlled by a master device (most often a microcontroller), which generates the serial clock (SCL), controls the bus direction, and generates the START and the STOP conditions.

The serial EEPROM is the slave. The serial EEPROM will be the bus transmitter during READ operations and when the serial EEPROM must acknowledge data transmitted by the master.

START and STOP bits control the bus activity. Operations must begin with a START bit and end with a STOP bit.

A START bit is when SDA transitions LOW while SCL is HIGH while observing the START set-up and hold time specifications.

A STOP bit is when SDA transitions HIGH while SCL is HIGH while observing the STOP set-up and hold time specifications.

Data is recognized as valid while SCL is high. The data on SDA must observe data in set-up and hold specifications before and after SCL is pulsed. There is only one bit of data for each SCL pulse.

Control Byte Requirements

After a START bit, each command begins with an 8 bit control byte sent by the master. This control byte has the following three primary functions before the data and/or word address information is loaded for all commands:

Identify the serial EEPROM as the slave addressed on the bus.

Select the specific serial EEPROM or the internal memory block on the bus. There may be up to 8 serial EEPROMs on the bus)

Select the READ or WRITE function for the next command transmitted by the master.

The diagram of a control byte (not including the START bit) is shown below:

1	0	1	0	A2	A1	A0	X
I²C Slave Address				Chip or Block Select			Read or Write bit

Control Bits 1-4 are the Slave Address Bits (Must be 1010 for Memory)

Since there is not a chip select pin, the part is selected by a four bit code in the control byte to identify the type of product. The four bit code was established by Philips for the I²C protocol. A 1010 code identifies the slave device as a Serial EEPROM. The Serial EEPROM will remain in stand-by until the 1010 code is transmitted on the bus. Other non Serial EEPROM slave devices will not respond to the 1010 code on the bus.

Control Bits 5-7 are the 1 of 8 Chip or Block Address Select Bits

The next three control bits are utilized for the chip selection or internal block selection. The standard I²C protocol was developed to allow up to 16K bits of memory to be selected. This could be accomplished by accessing a combination of devices or blocks within a device, as shown in the table on the following page:

© 1993 Microchip Technology Inc.

Device	K bits Density	Internal Blocks	A0	A1	A2	Bus Access Devices
24LC01B, 24C01,85C72	1	1	H or L	H or L	H or L	Up to 8 devices
24LC02B, 24C02,85C82	2	1	H or L	H or L	H or L	Up to 8 devices
24LC04B, 24C04,85C92	4	2	X	H or L	H or L	Up to 4 devices
24LC08B	8	4	X	X	H or L	Up to 2 devices
24LC16B	16	8	X	X	X	Only 1 device

X= NOT USED. This pin must be tied to VSS or VDD. It is not recommended to FLOAT these pins since there may be test modes accessed to these pins via a high voltage signal.

These three bits for this select must match the hardware conditions (IF ANY ARE USED) of the external A0, A1, and A2 pins or the internal block selects.

With this selection scheme, devices from 2K to 16K are software compatible. For example, four 2K devices or one 8K device could be connected to the bus with the same software.

The A0, A1, and A2 signals are the same for the 1K and 2K products. The A7 bit for the 1K product is a DON'T CARE.

The A0, A1, and A2 pins are not commonly used today in the industry with the advent of the density evolution up to the I²C protocol limit of 16K bits.

Control Bit 8 Operation Code

If this bit is a "1" then the operation will be a READ

If this bit is a "0" then the operation will be a WRITE

After the control byte acknowledge bit is generated by the serial EEPROM, the master will send the appropriate word address and data information.

Acknowledge Requirements

The serial EEPROM must generate an acknowledge bit after receiving each byte segment in a command. The serial EEPROM will generate the acknowledge bit automatically after the master has transmitted all of the data for the segment.

To acknowledge the master, the serial EEPROM must pull the SDA line LOW during the entire HIGH period of the next clock generated by the master. During the READ operations, the master must acknowledge each data byte or the serial EEPROM will abort the READ operation and return to a stand-by mode waiting for the next START bit.

The attached 24LC16 timing diagrams illustrate the READ and WRITE operations.

MICROCHIP 2-WIRE DEFAULT CONDITIONS

As stated before, data sheets do not provide adequate information on basic operation. This lack of information forces each reader of the databook to make interpretations about the operating conditions. These readers have included other semiconductor circuit designers, which unfortunately leads to subtle compatibility problems. The part is designed to operate to the default of the circuit designer's interpretation. This next section details Microchip's default conditions to help the system engineer minimize "Trial and Error" prototyping and to increase the awareness of these default conditions.

Also, to improve corporate-wide compatibility, Microchip is standardizing their circuits on various product versions. Unless indicated otherwise, all references to default conditions are for the 24LCXX products, not the 24CXXXX products.

Power Up

READ, WRITE, and ERASE operations are valid 5 uS after VDD has ramped to the specified operating range.

PAGE WRITE by Product for Multiple BYTE WRITE Operation

The 24C01 and 24C02 have a 2 byte buffer.

The 24C04 has an 8 byte buffer.

The 24LC01 and 24LC02 have an 8 byte page.

The 24LC04, 24LC08, and 24LC16 have a 16 byte page.

The buffer will load bytes identically as the page loads bytes. The difference in the two modes is that the buffer will execute a WRITE of one byte per WRITE cycle in sequence. The page mode will execute all bytes loaded in one WRITE cycle in parallel.

8

© 1993 Microchip Technology Inc.

There are pages within blocks. For a 16 byte page product, the most significant 4 bits of the word address point to the page address and the least significant 4 bits point to the byte address within a page. For an 8 byte page product, the most significant 5 bits of the word address point to the page address and the least significant 3 bits point to the byte address within a page.

The number of bytes loaded in to the page is from one byte up to the page size. For example, three bytes can be loaded into the 16 byte page of the 24LC16. If during the loading of the fourth byte a STOP bit is received, the page will WRITE three bytes. The fourth byte will not be written since loading the fourth byte was not complete.

NOTE: New versions released in March 1993 will default to ABORTING the entire operation if a STOP bit is received in the middle of a byte while loading a page.

If more than 16 bytes are loaded in the page of a 16 byte page product, then the 17th byte will override the data loaded into the original first byte (the page data will wrap around WITHIN a page). Therefore, the system designer must take precautions to not WRITE over a page boundary during a multiple byte WRITE operation.

Bytes not changed in the page will NOT result in data corruption in the array. For example, If two bytes are loaded in to the 24LC16 page with the least significant word address bits of 0000 and then a STOP bit is transmitted. Bytes 1 and 2 in the array will have the data changed to the new page contents. Bytes 3 through 16 WILL NOT change.

The WRITE operation will not be executed until a STOP bit is transmitted.

At this point, the serial EEPROM is free from the bus since the actual WRITE function is self-timed. Therefore, the microcontroller interfacing to the serial EEPROM may perform other functions not associated to communication with the serial EEPROM during the self-timed WRITE operations.

Once the part is in the auto-ERASE mode, it will complete the ERASE/WRITE operation unless power is removed. STOP and START bits will be ignored.

READ

Once the Serial EEPROM is in a RANDOM READ operation, it can be placed into the sequential READ operation. If the master issues an acknowledge bit instead of a STOP bit, the Serial EEPROM will READ the next sequential 8 bits. The Serial will wait for the next bit command from the master. The sequential READ will continue as long as the master issues an acknowledge bit on the next clock cycle after the last bit is READ. The READ will continue from block to block and will wrap around if the last bit in the array is addressed. Again, this will continue until the master issues a STOP bit instead of an acknowledge bit.

While reading zeroes the master cannot pull SDA high to generate a STOP bit, since the serial EEPROM SDA pin is outputting a low. To recover from a fault during a READ, repeat 9 clocks with data floating high. Therefore, the acknowledge bit will not occur and the part will reset and return to stand-by.

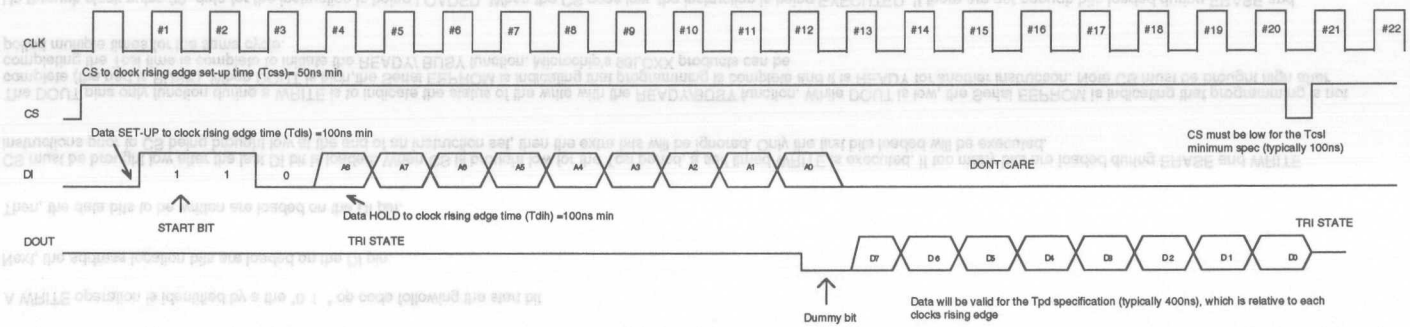
A START bit during an operation will cease the current operation and begin the next operation.

Author: *Steve Drehobl*
Memory Products Division

© 1993 Microchip Technology Inc.

3 WIRE BUS EXAMPLE

MICROCHIP 93LC66 READ CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR X8 OPERATION OF MICROCHIP'S 93LC66. INSTRUCTION LENGHTS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the RISING clock edge. Each parameter is typically 100ns.

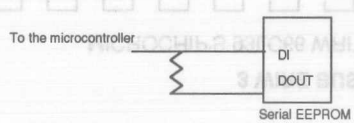
A Read operation is identified by the "1 0 " op-code following the start bit.

Next, the address location bits are loaded.

Then the address contents are clocked out on the rising clock pulse edge. Data will become valid on the DOUT pin per the specified Tpd time (typically 400ns) relative to the rising edge of the clock on the DOUT pin. Note, the first bit output will be a "dummy bit" with a logical zero state. This event is triggered by the clock rising edge of the last address bit for a duration of one clock pulse.

If the data from the current address is complete and the clock pulses continue, the data from the next address will be READ automatically as long as CS remains high. This is the SEQUENTIAL READ FUNCTION. READ operations will continue while clock pulses continue or until CS is brought low.

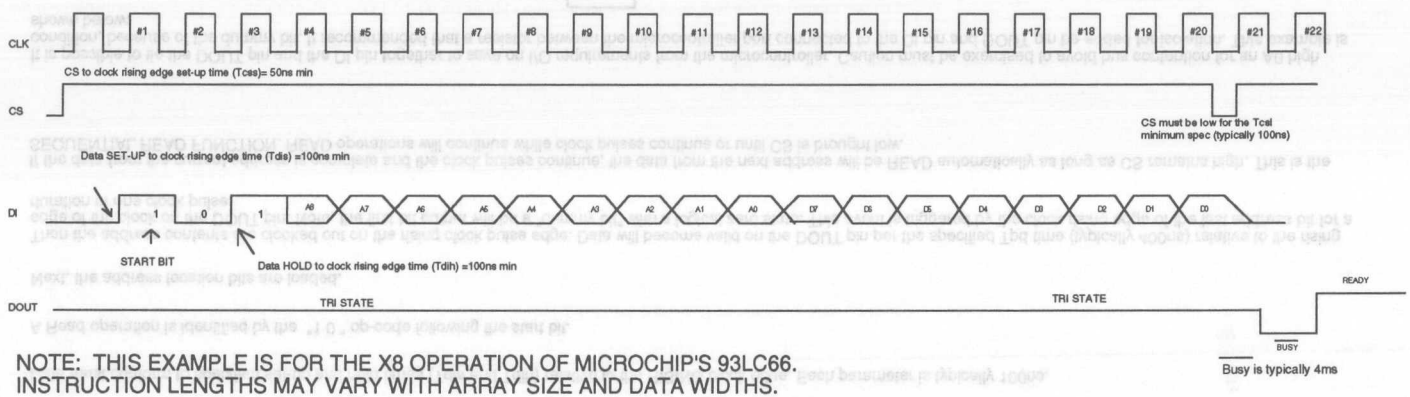
It is possible to tie the DOUT pin and the DI pin together to save on I/O requirements from the microcontroller. Caution must be exercised to avoid bus contention for an A0 high condition, because of the dummy bit. It recommended that a resistor between the microcontroller port connected to the DI pin and DOUT pin be added for isolation. This example is shown below:



© 1993 Microchip Technology Inc.

3 WIRE BUS EXAMPLE

MICROCHIP'S 93LC66 WRITE CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR THE X8 OPERATION OF MICROCHIP'S 93LC66. INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS.

Data must conform to specific set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

A WRITE operation is identified by a the "0 1 " op code following the start bit

Next, the address location bits are loaded on the DI pin.

Then, the data bits to be written are loaded on the DI pin.

CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcsl period, a self timed WRITE is executed. If too many bits are loaded during ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

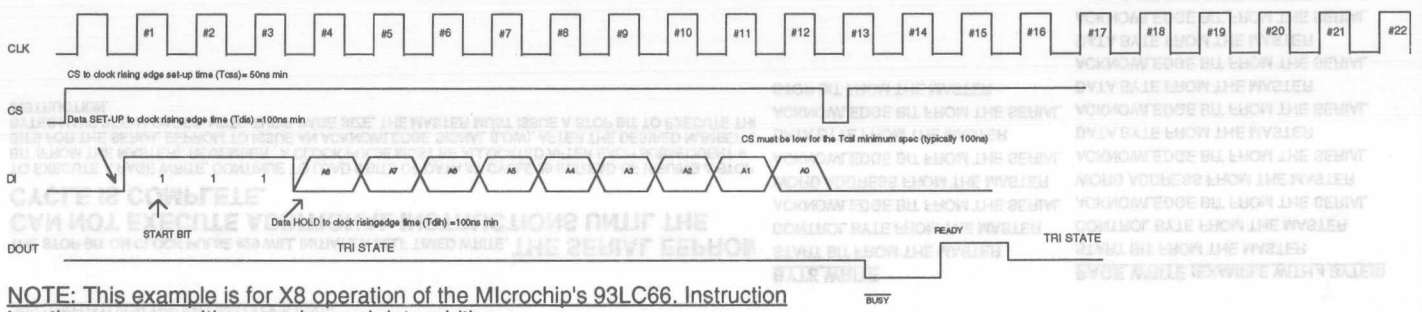
The DOUT pins only function during a WRITE is to indicate the status of the write with the READY/BUSY function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When DOUT is high,the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcsl time is complete to initiate the READY/ BUSY function. Microchip's 93LCXX products can be polled multiple times for the same cycle.

Up through clock pulse 20, data for the instruction is being LOADED. When the CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the Serial EEPROM will return to stand-by.

© 1993 Microchip Technology Inc.

3 WIRE BUS EXAMPLE

MICROCHIP 93LC66 ERASE CYCLE TIMING DIAGRAM



NOTE: This example is for X8 operation of the MIcrochip's 93LC66. Instruction lengths may vary with array size and data widths.

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

AN ERASE operation is identified by a "11 " two bit code that follows the start bit

Next, the address location bits are loaded on the DI pin.

THERE ARE NO DATA BITS TO LOAD. THE ADDRESS LOCATION LOADED WILL BE SET TO AN ERASE STATE OF "1".

CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcsl period, a self timed ERASE is executed. If too many bits are loaded during the ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

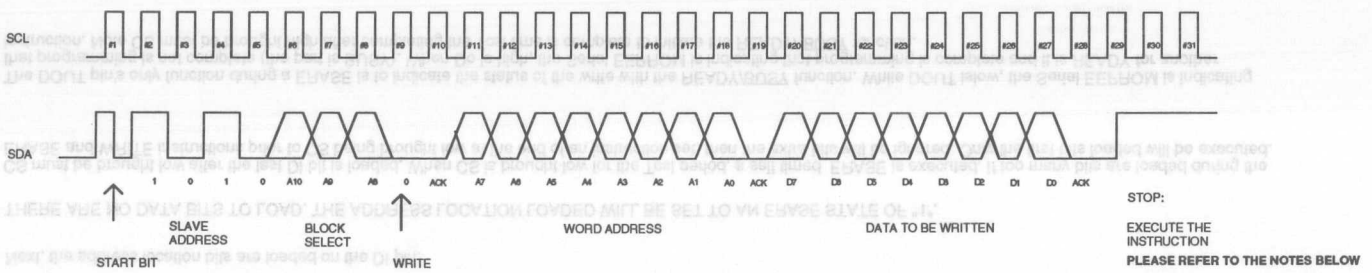
The DOUT pin's only function during a ERASE is to indicate the status of the write with the READY/BUSY function. While DOUT islow, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When Do is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcsl time is complete to initiate the READY/ BUSY function.

Up through clock pulse 12, the address for the instruction is being LOADED. When CS goes low, the instruction is being EXECUTED.If there are not enough bits loaded during the ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the serial EEPROM will return to stand-by.

© 1993 Microchip Technology Inc.

2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 BYTE WRITE CYCLE TIMING DIAGRAM



NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET -UP TIME 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME C 0NS (THD:DAT) FOR THE FALLING CLOCK EDGE.

THE STOP BIT ON CLOCK PULSE #29 WILL INITIATE A SELF TIMED WRITE. **THE SERIAL EEPROM CAN NOT EXECUTE ADDITIONAL INSTRUCTIONS UNTIL THE CYCLE IS COMPLETE.**

TO EXECUTE A PAGE WRITE, CONTINUE TO LOAD 8 BITS OF DATA AT CYCLE 29 INSTEAD OF ISSUING A STOP BIT (FROM THE MASTER). REMEMBER , A CLOCK PULSE MUST BE ALLOCATED AFTER EACH SUBSEQUENT 8 BITS FOR THE SERIAL EEPROM TO ISSUE AN ACKNOWLEDGE SIGNAL (LOW). AFTER THE DESIRED NUMBER C BYTES HAVE BEEN LOADED, UPTO THEIR PAGE SIZE, THE MASTER MUST ISSUE A STOP BIT TO EXECUTE THI INSTRUCTION.

THE SEQUENCE OF EACH WRITE COMMAND AND THE MASTER/SERIAL DIRECTION OF THE COMMUNICATION IS SHOWN BELOW :

BYTE WRITE

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER

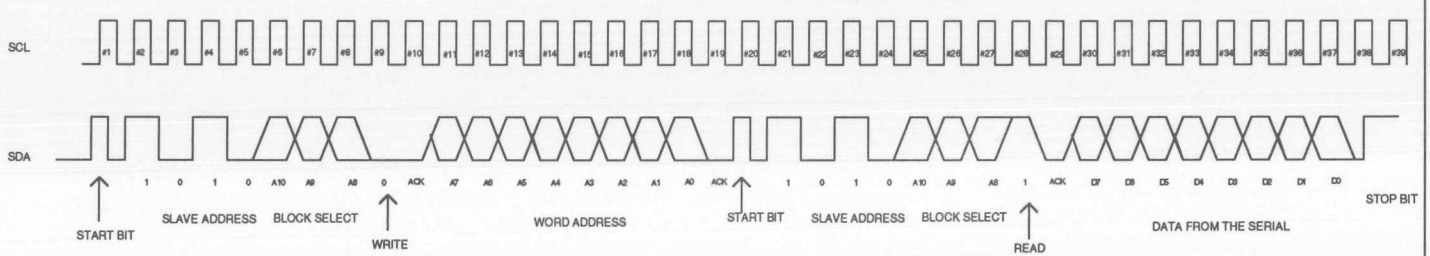
PAGE WRITE (EXAMPLE WITH 4 BYTES)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER

© 1993 Microchip Technology Inc.

2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 READ CYCLE TIMING DIAGRAM



NOTE : THE FIRST 19 CLOCK PULSES OF THE WRITE COMMAND ARE IDENTICAL TO THE FIRST 19 CLOCK PULSES IN THE READ COMMAND. EVEN THE 9TH CLOCK PULSE IS A WRITE BIT "0" TO TRANSMIT TO THE SERIAL EEPROM THE DESIRED WORD ADDRESS.

THE READ COMMAND HAS TWO START BITS. THIS IS FOR THE RANDOM READ COMMAND.

AS SHOWN ON THE PREVIOUS PAGES, IF A READ IS DESIRED FROM A CURRENT ADDRESS THEN THE FIRST 19 CLOCK PULSES ARE NOT REQUIRED. THEREFORE, THE FIRST START BIT IS AT CLOCK PULSE #20. THIS IS ONLY FOR THE CURRENT ADDRESS READ COMMAND

ANOTHER USEFUL READ COMMAND IS THE SEQUENTIAL READ COMMAND. THE SEQUENTIAL READ COMMAND IS THE SAME AS THE RANDOM READ COMMAND; HOWEVER, THE MASTER MUST ISSUE AN ACKNOWLEDGE BIT INSTEAD OF THE STOP BIT AS SHOWN FOR CLOCK PULSE #38. THIS SIGNALS THE SERIAL TO READ THE DATA FROM THE NEXT SEQUENTIAL ADDRESS. THE MASTER MUST CONTINUE TO ACKNOWLEDGE EACH BYTE RECEIVED UNTIL THE MASTER ISSUES A STOP BIT.

NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET -UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD:DAT) FOR THE FALLING CLOCK EDGE.

READ (FROM A RANDOM ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (FROM THE CURRENT ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (Sequential READ of 3 bytes)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R/W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

© 1993 Microchip Technology Inc.

NOTES:



© 1993 Microchip Technology Inc.

MICROCHIP

AN537

Everything a System Engineer Needs to Know About Serial EEPROM Endurance

The term "endurance" has become a confusing parameter for both users and manufacturers of EEPROM products. This is largely because many semiconductor vendors treat this important application-dependent reliability parameter as a vague specmanship topic. As a result, the system engineer often designs without proper reliability information or under-utilizes the EEPROM as an effective solution.

Endurance (the number of times an EEPROM cell can be erased and rewritten without corrupting data) is a measure of the device's reliability, not its parametric performance. As such, endurance is not achieved by somehow making EEPROM devices more durable or robust to extend the life of the intrinsic erase/write cycle, but rather by reducing their defect-density failure rates. This has a direct impact on the design engineer characterizing EEPROM memory needs for an application and evaluating components from various manufacturers. The system design engineer needs to understand not only the relationship between the application, expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

This tutorial volume is intended to clarify some of the issues in the industry and provide a tool for the system design engineer, the system reliability engineer, and the component engineer to determine EEPROM reliability and understanding how to apply it to actual application requirements. It will examine four main areas:

- CMOS floating gate memory cell operation and characteristics
- Significant process and design interactions and endurance characterization variables
- Common misinterpretations of endurance
- Determining some real world application reliability requirements

EEPROM MEMORY CELL OPERATION AND CHARACTERISTICS

In discussing endurance characteristics of EEPROMs, it's important to review how these components operate, and why and how they fail. Figure 1 illustrates a CMOS floating gate EEPROM cell, including voltage conditions for READ, ERASE, and WRITE operations. To erase or write, the row select transistor must have the relatively high potential of 20V. This voltage is internally generated on chip by a charge pump, with the only external voltage required being V_{DD}. The only difference between an ERASE and a WRITE is the direction of the applied field potential relative to the polysilicon floating gate.

When 20V is applied to the polysilicon memory cell gate and 0V is applied to the bit line drain (column), electrons tunnel from the substrate through the 90-angstrom Tunnel Dielectric (TD) oxide to the polysilicon floating gate until the polysilicon floating gate is saturated with charge. The cell is now at an ERASE state of "1". When 0V is applied to the polysilicon memory cell gate and 20V is applied to the bit line drain (column), electrons tunnel from the polysilicon floating gate through the TD oxide to the substrate. The cell then is at a WRITE state of "0". This sequence of the transfer of charge onto the floating gate (ERASE) and the electrical removal of that charge from the floating gate (WRITE) is one ERASE/ WRITE cycle, or "E/W cycle."

The field (applied voltage to an oxide thickness) across the tunneling path created by the 20V potential is extremely high in order to transfer the electrons. Over the cell's "application time," as measured by E/W cycles, the EEPROM cell begins to wear out due to the field stress. The EEPROM cell wears out as the number of cycles increase resulting in the voltage margin between the ERASE and WRITE states decreasing until finally there is not enough margin for the EEPROM sense amp to detect a difference in the two states during a READ. Failure is defined as when the sense amp can no longer reliably differentiate logic state changes.

Figure 2 (single cell EEPROM endurance characteristics) illustrates that the intrinsic wear out point for a normal cell with specified dimensions and electrical characteristics is very acceptable, in excess of 2 million E/W cycles. Failures at lower cycles are due mostly to very small defects or imperfections in the oxide or silicon-to-oxide interface.

8

© 1992 Microchip Technology Inc.

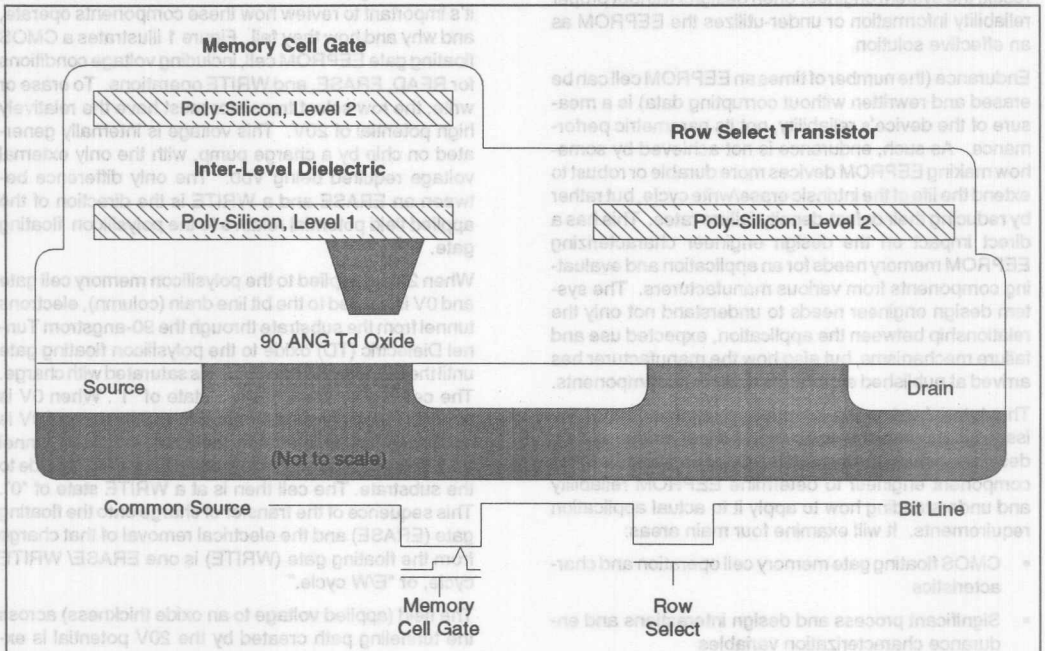
Serial EEPROM Endurance

A key point to remember is that most failures occurring at less than 2 million E/W cycles are due to the number of defects per a given area (defect density dependent.) Thus high EEPROM endurance reliability is achieved by reducing the defect density failure rates, not by increasing the number of intrinsic cycles in the cell's operational design.

Error correction circuits are design techniques commonly used by EEPROM manufacturers to increase endurance by reducing the failure rate caused by single bit failures. These circuits are transparent to the user. One typical scheme is using 4 bits of error correction for every 8 "real" bits (one byte). In this scheme, one bit failure in the byte is correctable, while if two bits within the byte fail, the byte is not correctable.

Another error correction scheme is to use one "parity" bit for every "cell." Here both EEPROM cells must fail to result in a bit fail.

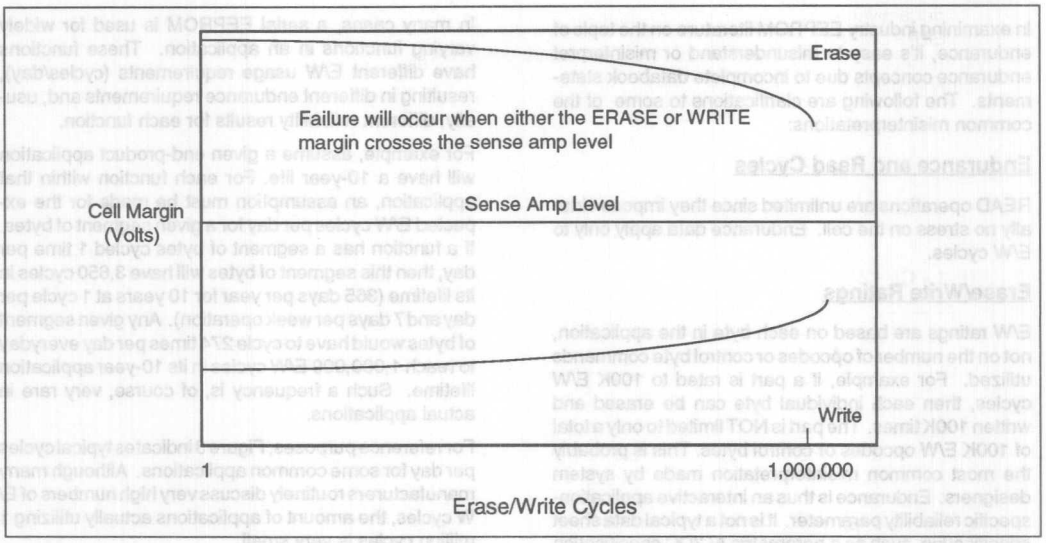
FIGURE 1 - CMOS FLOATING GATE EEPROM CELL



	Read	Write	Erase	Standby
Bit Line	1.6 volts	18.0 volts	0.0 volts	0.0 volts
Row Select Gate	5.0 volts	20.0 volts	20.0 volts	0.0 volts
Memory Cell Gate	5.0 volts	0.0 volts	20.0 volts	0.0 volts
Common Source	0.0 volts	Float	0.0 volts	0.0 volts

© 1992 Microchip Technology Inc.

FIGURE 2 - SINGLE EEPROM CELL ENDURANCE CHARACTERISTICS



PROCESS AND DESIGN VARIABLES AFFECTING ENDURANCE

There are many subtle process and design variables that have a strong impact on endurance. These interacting variables will play very different roles depending on the different process technologies of various semiconductor manufacturers.

The primary interaction is the amount of TIME at the HIGH VOLTAGES that is ultimately applied to the cell. A finite amount of time at finite voltages are required to achieve "optimal" ERASE and WRITE thresholds. If the time is too short and the voltage is too low, the EEPROM will not program to the proper threshold. Also, if the programming ramp time is too fast and the voltage is too high, the EEPROM's endurance will be reduced. Unfortunately, there is most often a trade-off between fast reliable programming performance or high endurance reliability. Some of the significant process and design variables are shown below and their impact on programming performance and endurance performance.

PARAMETER	PROGRAMMING	ENDURANCE
Internal High Voltages	HIGHER = Faster Programming	LOWER = Increased Endurance
Internal High Voltage Ramp Rate	FASTER = Faster Programming	SLOWER = Increased Endurance
Programming Time	LONGER = Improved Voltage Margin on the Cell	SHORTER = Less Oxide Stress for Increased Reliability
TD Oxide thickness	THICKER = Slower Programming	THINNER = Reduced Endurance
Temperature	LOWER = Faster Programming	LOWER = Increased Endurance

8

© 1992 Microchip Technology Inc.

COMMON MISINTERPRETATIONS

In examining industry EEPROM literature on the topic of endurance, it's easy to misunderstand or misinterpret endurance concepts due to incomplete databook statements. The following are clarifications to some of the common misinterpretations:

Endurance and Read Cycles

READ operations are unlimited since they impose virtually no stress on the cell. Endurance data apply only to E/W cycles.

Erase/Write Ratings

E/W ratings are based on each byte in the application, not on the number of opcodes or control byte commands utilized. For example, if a part is rated to 100K E/W cycles, then each individual byte can be erased and written 100K times. The part is NOT limited to only a total of 100K E/W opcodes or control bytes. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement.

Cycles/Day

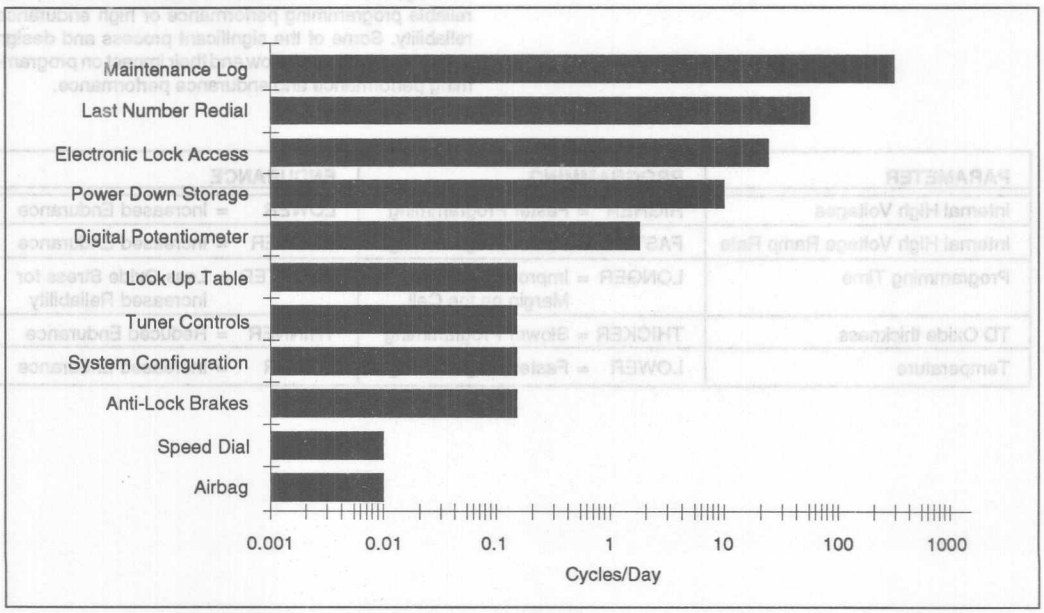
In many cases, a serial EEPROM is used for widely varying functions in an application. These functions have different E/W usage requirements (cycles/day), resulting in different endurance requirements and, usually, different reliability results for each function.

For example, assume a given end-product application will have a 10-year life. For each function within that application, an assumption must be made for the expected E/W cycles per day for a given segment of bytes. If a function has a segment of bytes cycled 1 time per day, then this segment of bytes will have 3,650 cycles in its lifetime (365 days per year for 10 years at 1 cycle per day and 7 days per week operation). Any given segment of bytes would have to cycle 274 times per day everyday to reach 1,000,000 E/W cycles in its 10-year application lifetime. Such a frequency is, of course, very rare in actual applications.

For reference purposes, Figure 3 indicates typical cycles per day for some common applications. Although many manufacturers routinely discuss very high numbers of E/W cycles, the amount of applications actually utilizing 1 million cycles is very small.

A further and very important incorrect assumption often made is that ALL bits in an application need the same number of cycles and endurance ratings. In most applications, however, functions that require a high

FIGURE 3 - TYPICAL SERIAL EEPROM E/W CYCLES/DAY BY APPLICATION



© 1992 Microchip Technology Inc.

number of E/W cycles per day require only a small number of bits. Last number redial in a telephone, for example, consumes many E/W cycles per day, but utilizes only a few bytes for this function. By contrast, speed dial storage in that same telephone consumes only a fraction of E/W cycles per day, but requires a relatively large segment of bits to accommodate the many speed dial options. In such an application, the same serial EEPROM normally performs both functions at different address locations.

ENDURANCE DATA FROM THE CUSTOMER'S PERSPECTIVE

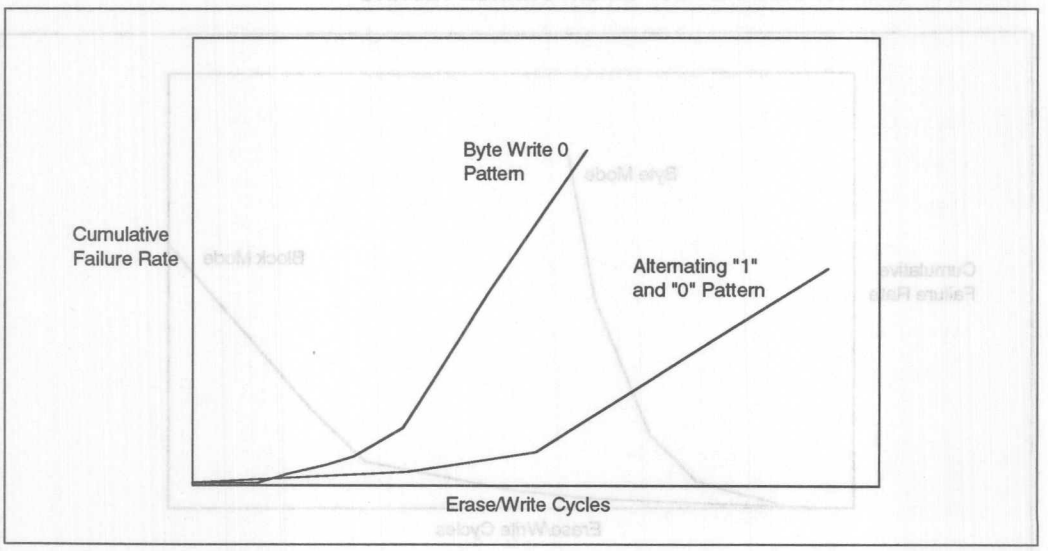
Unfortunately, an industry standard for an endurance test method has yet to be adopted. Since endurance data is not baselined, the process of evaluating endurance becomes that much more complicated for the system designer and reliability engineer.

It is not uncommon for customers to request endurance data from many semiconductor vendors. All vendors would be expected to comment that they experience a low failure rate through 100K E/W cycles. While this can be a true statement, it can also be a very incomplete statement. It is extremely doubtful that all vendors test their components to the same conditions. Yet the variables within endurance testing are extremely significant. Small differences in text protocol can have enormous differences. Pattern, cycling mode, temperature and array size, for example, are the most significant testing variables.

First, memory cell failure rates are defect density driven up to the intrinsic wear out point. Existing defects in a cell, while not causing failure initially, are stressed during every transfer of electrons through the TD oxide until they eventually cause cell failure. Worst case testing would be to erase and write each bit, which is what a write all "0"s pattern with an auto-erase of "1" routine will perform. Indeed, this write all "0"s test pattern will produce very different results than a checkerboard test pattern of alternating "1"s and "0"s within a byte, since cells are changed more often writing all "0"s than in an alternating "1" and "0" write pattern. The resultant failure rate differences are indicated on the pattern effect graph in Figure 4.

In actual use, however, a system will experience a random pattern much more like the alternating "1"s and "0"s pattern than the more stressful all "0"s pattern. The key point for system designers is to determine how accurate a test routine has been used to determine a particular manufacturer's endurance data, and make the appropriate judgement on that part's expected endurance in the application.

FIGURE 4 - PATTERN EFFECT ON ENDURANCE TESTING



8

© 1992 Microchip Technology Inc.

Second, the cycling mode graph in Figure 5 indicates that significantly different results can be achieved in endurance testing using a block cycle mode than using a byte cycle mode. The block mode is commonly used by manufacturers to "speed up" the endurance characterization process. However, endurance results usually will appear much better for the block mode than the byte mode, due to the high voltage variables discussed earlier. The reason is that the voltage ramp rate is significantly SLOWER, the high voltages are slightly lower, thus less stressful for block cycling since the capacitive load of the entire array is on the high voltage charge pump. The capacitive load is much lower with a single row or byte, which thus has a significantly faster ramp rate. Also, there is not a polysilicon to polysilicon stress for adjacent cells since all cells are at the same potential. Most often, these factors combine to yield lower failure rates for block cycling than for byte cycling. Again, the test conditions must match the system conditions.

Finally, increasing temperature also increases stress on the cell. Microchip's endurance characterization data indicates that increasing temperature adds an activation energy (Ea) of 0.12eV on the cell. From a 25°C to 85°C ambient the acceleration factor is approximately 2.1. Therefore, the higher the temperature, the higher the stress. These results will vary significantly with each EEPROM manufacturer.

RECOMMENDED EEPROM ENDURANCE TESTING

Microchip believes that EEPROM components should be endurance tested to reflect system conditions. Therefore, units are cycled to an alternating ONE and ZERO pattern (checkerboard), then to an alternating ZERO and ONE pattern (inverse checkerboard).

Again, since endurance characterization data indicates that a random single bit fail is the primary first order failure mode, endurance is defect density (def/cm2 or segment of bit size) dependent. Therefore an expected failure rate range by density can be established.

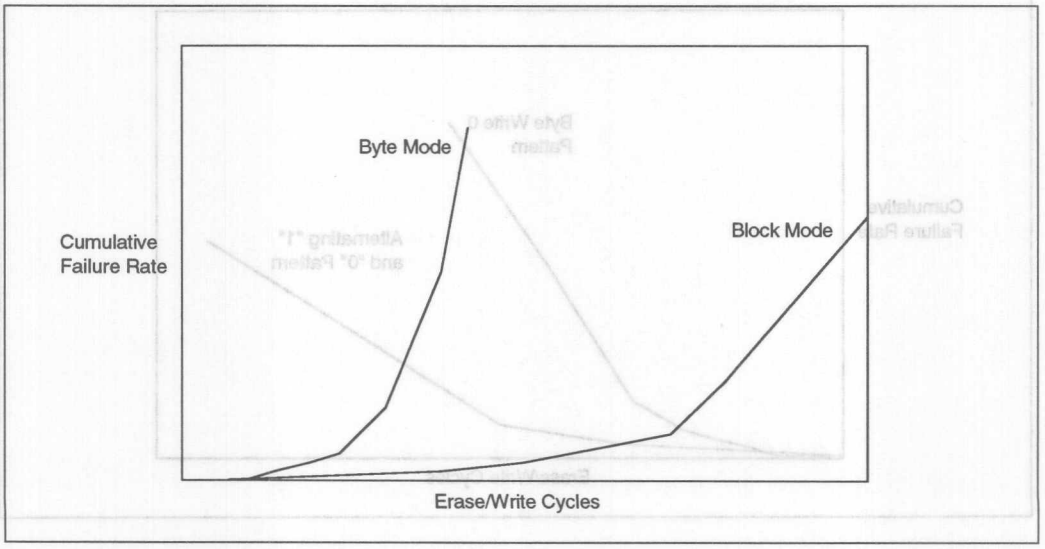
DETERMINING THE RELIABILITY CALCULATIONS

There are three primary components for the system design engineers to use in determining the endurance reliability required for a defect density limited application. These three components are:

- Erase/write cycles/day estimated for the function.
- The number of bits in the function (or segment size).
- Case operating temperature of the Serial EEPROM.

Let's look at three typical examples utilizing the above information to predict a cumulative failure rate at different points in a system lifetime. Please note that Industry endurance perceptions have improved from a very high (> 2%) failure rate expectation to a very low actual PPM level failure rate in the past few years.

FIGURE 5 - CYCLING MODE EFFECT ON ENDURANCE TESTING



© 1992 Microchip Technology Inc.

EXAMPLE #1 - AUTOMOBILE ELECTRONIC COMPASS

A 16-byte (128-bits) segment from a 2K array stores data every time the power is turned off in an automobile electronic compass. The design engineers expect the system to power down an average of 5 times/day over a 10-year life in an 85°C case temperature environment. The projected Microchip cumulative failure rate under these conditions at the 10 year point is less than 8 PPM or 0.0008% failures in 10 years. The cumulative PPM failure rate graph is referenced on Figure 6.

EXAMPLE #2 FULL-FEATURED TELEPHONE

An 8-byte (64-bit) segment is used to store the last number redial on a stationary full feature phone operating in a room temperature environment. A 12K bit (12,288-bits) segment is used for storage of speed dial numbers on the same phone. This application has two major functions and therefore it will have two separate failure rate calculations.

The last number redial function is expected to be utilized an average of 20 times/day over a 10 year life. Each speed dial is expected to be updated an average of 0.1 times/day over a 10 year life.

Figure 7, the cumulative failure rate graph for these two conditions indicates an extremely low failure rate in the projected 10 year lifetime. The failure rate begins beyond 20 years as shown on the attached cumulative PPM failure rate graph.

EXAMPLE #3 - LASER PRINTER

A serial EEPROM could have many functions in a laser printer. A function that would likely require the most cycles/day is the maintenance log storage of the pages printed (estimated to be 100 cycles/day). Three bytes are utilized to store this number. The case temperature environment is estimated to be between 55°C and 85°C

The high number of cycles at an extreme temperature of 85°C indicate a failure rate of less than 1200 PPM through the first 5 years and 2600 PPM through the first 10 years.

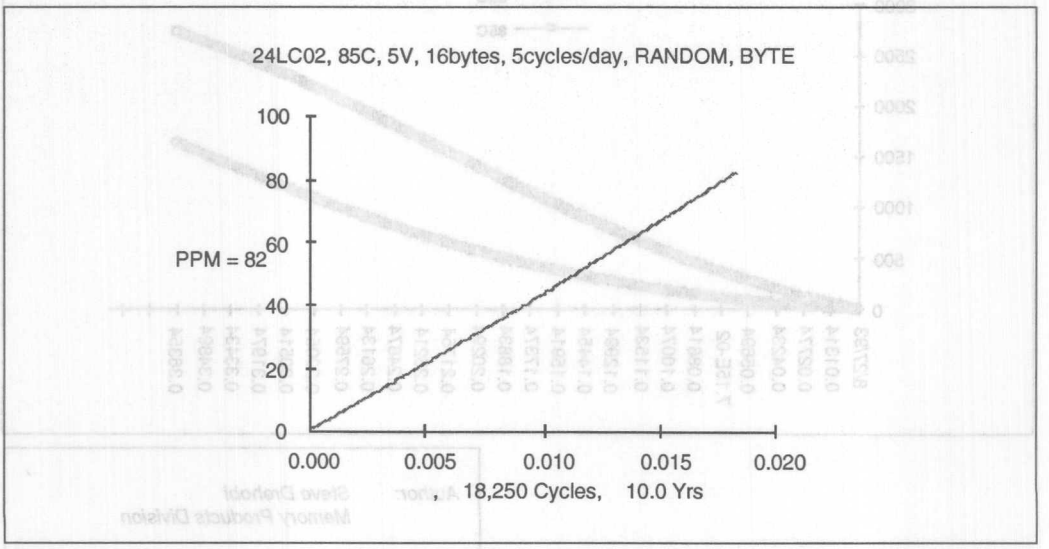
This failure rate can be dramatically reduced if the operating temperature is reduced to 55°C. The same 5 and 10 year PPM levels are reduced to 450 PPM and 1600 PPM at 55°C.

These failure rates are illustrated on Figure 8.

SUMMARY

Microchip Technology Inc. has recognized that increasing reliability in serial EEPROMs through increased endurance is not a function of extending the life of the intrinsic erase/write cycle, but depends on reducing defect-density failure rates. Design engineers characterizing EEPROM memory needs for an application and evaluating EEPROM components from various manufacturers need to understand not only the relationship between the application and expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

FIGURE 6 - EXAMPLE #1: THE AUTOMOBILE COMPASS



8

© 1992 Microchip Technology Inc.

Microchip has developed a program to calculate the cumulative PPM failure rates for specific system conditions, as shown on the previous pages. This program is being transferred onto menu driven DOS floppy disks. These disks will be available in December 1992 to the system designers, reliability engineers, and component engineers to project endurance failure rates for specific system conditions. These disks will be periodically updated to reflect Microchip's most recent endurance data.

FIGURE 7 - EXAMPLE #2: THE FULL FEATURED PHONE

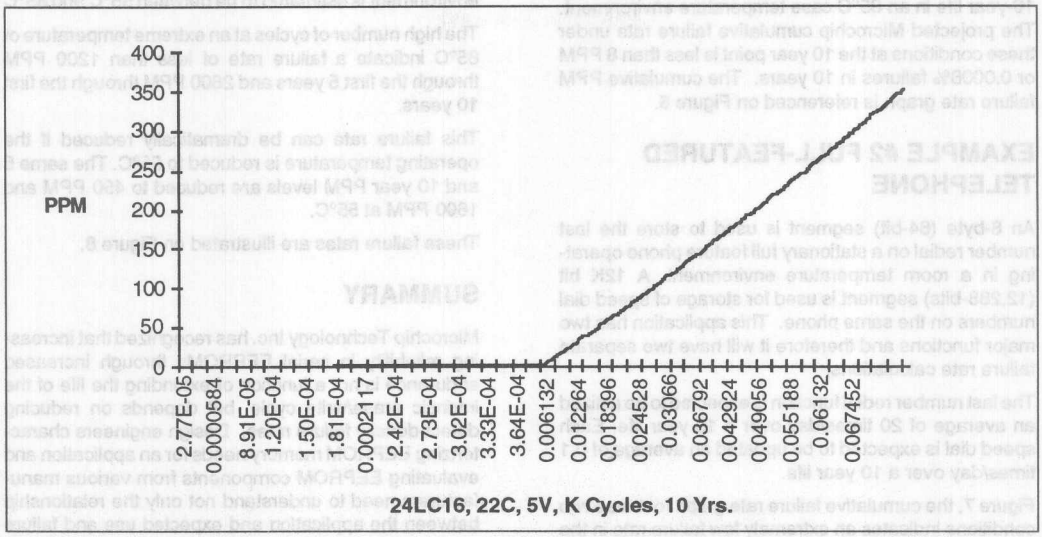
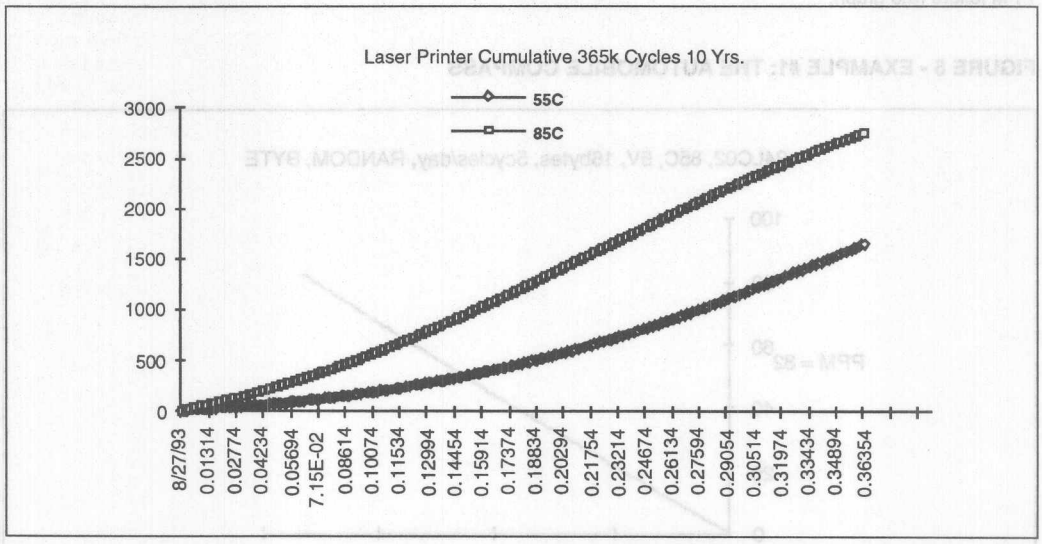


FIGURE 8 - EXAMPLE #3: THE LASER PRINTER



Author: *Steve Drehobl*
Memory Products Division

© 1992 Microchip Technology Inc.

MICROCHIP

AN562

Using the Microchip Endurance Predictive Software

INTRODUCTION

Endurance, as it applies to non-volatile memory, refers to the number of times an individual memory cell can be erased and/or written (some architectures do not erase before every write). Advances in process technology have made it possible to increase these limits and for Microchip to offer new concepts - Total Endurance™ and a split architectural design for variable endurance. These concepts lead to more reliable products with more bits per dice, such as the 24C32 and 24C65.

TOTAL ENDURANCE™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate (for each byte in the application not on the number of opcodes or control byte commands) and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. These studies led to the concept of using the computer to predict the theoretical wear out of the floating gate and ultimately to project the point in time of a product's life cycle when the first non-volatile memory bit or periphery failure should occur. After many man-years of data collecting, predicting and verifying the results, Microchip feels confident in publishing and offering for the general technical community this predictive model in the form of IBM® PC-compatible software. Microchip has a patent pending on this predictive mathematical model.

TOTAL ENDURANCE PREDICTIVE SOFTWARE

The predictive software described here originally was being developed as a tool for determining endurance levels of Microchip non-volatile devices. Upon seeing the potential of the software as a design aid, it was decided to make a version that could be purchased by the engineering community. The benefit gained from this software is the ability to predict endurance capability of Microchip's EEPROM devices under various operating conditions. Prior to this tool becoming available, the only way to assemble this type of data would be to do extensive life testing in the target system. It should be noted that this predictive model applies only to Microchip Technology Inc. non-volatile devices.

The program uses an iterative statistical model developed by Microchip Technology Inc. physicists. The model was first used in a DOS-based text program as a proof of concept and for developing the exhaustive database needed for such a tool (included on the program disk as enddos.exe). This model was then imported to a Windows™-based software package with full GUI capabilities and all the normal cut, paste, print, viewing properties. The model actually operates as a mathematical function which is called from within the Windows Visual Basic shell and is passed all of the pertinent operational, process, and device information. The model then, after calculating the essential data points, returns this information to the main program to be formatted and displayed both textually and graphically.

Applying the predictive data to the high endurance block of the 24C65, using the 24LC04 which has similar characteristics, and assuming the following:

- a five-year life
- an expected E/W cycles of 10 times per day
- a function of 11 bytes

8

© 1993 Microchip Technology Inc.

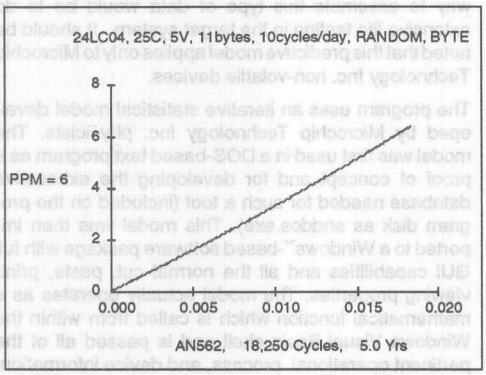
Operational specifications:

Device	24C65 (24LC04B)
Voltage	5
Temperature	25 C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4K HE block with 1 M E/W cycles typical, in this application, should yield the following results:

FIT	1.0	PPM	6
Time	5.0	Write cycles	18,250

FIGURE 1



The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance, see the related application notes AN536 and AN537 contained elsewhere in this volume. All operation parameters, along with the process technology, effect the effective endurance of a non-volatile device. The voltage, temperature, cycles per, bytes per cycle, and even the number of times written per day (time between write cycles) all have an effect on the oxide breakdown or periphery failure rate of a particular non-volatile process.

Endurance is not a well-defined concept within the semiconductor industry. The number of erase/write cycles which a particular EEPROM can endure is dependent not only upon the design of the device but also upon the application environment in which it is used. Therefore, blanket claims such as "1 million erase/write cycles typical" can only be validated based upon the specific parameters of each application. Yet until now, there has been no tool available for predicting the endurance of a particular EEPROM device within a set of application parameters. Trade-off analysis can be painfully time-consuming and only marginally accurate without specific knowledge of the behavior of the device under different conditions of use.

The Microchip Total Endurance Software allows the designer to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, PPM and FIT rates, and years of use in order to optimize the system and accurately predict product lifetime and reliability.

The following is an example using the Endurance Software to aid in the design of an electronic phone book/auto-dialer:

The auto-dialer may have new numbers added or changed several times per day; but how can the manufacturer specify the life of the unit, and at what rate of update of the phone numbers? First, the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacturer is willing to live with a 0.1% failure rate (1,000 PPM) after 10 years of use, then we have almost enough information to verify whether we are in the ball park given the physics of the EEPROM device which will store the numbers. We also need to know the operating voltage and temperature of the application; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the LCD display will function: 0°C to 70°C. End-of-life voltage for the battery is approximately 2.0V; assuming that the ASIC or microcontroller in the application will operate down to 2.5V, the EEPROM also has a 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6K byte requirement for the Serial EEPROM. Because 1.6K bytes is equal to 12.8K bits, a 16K bit 2-wire Serial EEPROM will more than suffice. Specifically, Microchip's 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

Device	24LC16B
Voltage	2.5V - 3.3V
Temperature	0°C to 70°C (55°C typical)
Cycles per day	50
Bytes per cycle	16
Application life	10 years

© 1993 Microchip Technology Inc.

Once these values are entered into the Total Endurance program, it outputs the following:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	50
App. Life (Yrs)	10
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
FIT	21.0
PPM	1,842
Time	10.0
Write cycles	182,500

Both of the lists above were copied directly from the Total Endurance program output to the Microsoft® Windows clipboard and pasted into this document (the Total Endurance program has a handy menu click to make this easy).

Unfortunately for our designer, the desired 0.1% failure rate has almost doubled to 0.18% (1842 PPM). But fortunately for the designer, the Total Endurance program makes trade-off analysis very simple and fast. At this point there are at least three options: (1) live with almost 2000 PPM, or (2) look at the endurance plot and check whether there is a reasonable number of E/W cycles which will provide a 1000 PPM failure rate, or (3) specify a PPM rate to the Total Endurance program and let it crank out the number of cycles it will take.

Below is the endurance plot, again pasted directly from the Total Endurance program:

You can see that by reducing the number of cycles from the 182,500 which resulted from our first trial to about 100,000, we can achieve a PPM rate of about 1000 (0.1%). But how does 100,000 cycles translate into application life or cycles per day?

By switching the Total Endurance program mode to a PPM request mode instead of application life mode, we can query the program for this information. Let's ask it for the application life of the product given a 1000 PPM failure rate. Here are the results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	50
PPM Level (Yrs)	1000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
PPM	1,000
Time	5.97
Write cycles	109,000

Now we have some more options: (1) specify the product life at 5 years or (2) trade-off other parameters of the application such as voltage or temperature, or (3) decide which is more important – a 10-year product lifetime, or the ability to change 50 numbers every single day. Maybe this analysis has caused our designer to re-evaluate the 50 cycle-per-day requirement. Will the user really change or add that many numbers per day – half of the unit's total capacity? Maybe 20 or even 10 is a more practical figure. Realistically, a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 per day gives us the following results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55
Bytes/Cycle	16
E/W	20
PPM Level (Yrs)	1000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

© 1993 Microchip Technology Inc.

Device Data:	Output Parameters
PPM	1,000
Time	14.93
Write cycles	109,000

Wow! Reducing the number of cycles per day not only brought us back to a 10-year life, it gave us some margin on that, too. Keeping all the other parameters the same and forcing a 10-year lifetime gives us the following final results:

Device Data:	Output Parameters
FIT	7.1
PPM	625
Time	10.0
Write cycles	73,000

The new PPM rate of 625 gives our triumphant designer more than 30% margin on his PPM target of 1000.

This example shows the significant reduction in time for design trade-off analysis and time-to-market which can be achieved with a useful tool like the Microchip Total Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by providing known quantities and readily accessible handles to modify those quantities in the trade-off analysis. This tool can literally reduce weeks of effort into a few minutes of point and click.

Authors: *Peter Sorrells*
Memory Products Division

Richard J. Fisher
Memory Products Division

© 1993 Microchip Technology Inc.

Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54

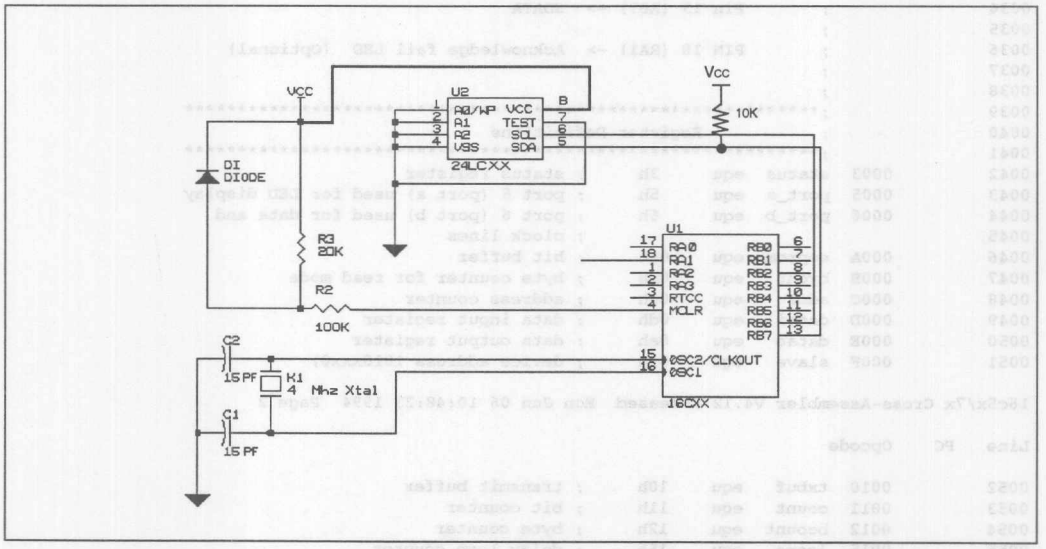
INTRODUCTION

The 24LCXXB Serial EEPROMs from Microchip Technology are I²C™ compatible, will operate in the standard 100kHz and the 400kHz Fast Mode. When using any serial protocol with a general microcontroller that does not have a dedicated protocol specific serial port, the designer must generate the specific code routines to accomplish the several memory access functions that the microcontroller will perform. The PIC16CXX products from Microchip are both versatile and efficient to program. This application note is a series of standalone programs to perform the basic I²C interface functions on a PIC16C54 running at 4MHz in XT mode. This configuration will provide a 60 kHz serial bus rate. At this speed, the interface does not meet the 100kHz specification. If a higher clock rate is desired, the HS crystal oscillator must be used, which has a maximum frequency of 20MHz. NOTE THAT THE TIMING ROUTINES MUST BE REWRITTEN TO RUN AT THESE FASTER CLOCK RATES. The user must consult all applicable data sheets for design details. These programs have been fully tested and are being made available for general use. Figure 1 demonstrates the tested configuration of PIC16C54 and the 24LCXXB device.

This application note includes the following programs.

- 2-Wire Byte Read
- 2-Wire Byte Write
- 2-Wire Byte Write with Data Polling
- 2-Wire Page Write
- 2-Wire Sequential Read

FIGURE 1 - TESTED CONFIGURATION OF THE PIC16C54 AND THE 24LCXXB DEVICE



Author: *Bruce Negley*
Memory Products Division

© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:23 1994  Page 1

Line   PC    Opcode

0001                 LIST P=16C54,c=132
0002            ;*****************************************************************
0003            ;       2-Wire Byte Read Program (118 bytes)
0004            ;
0005            ;       This program demonstrates how to interface a
0006            ;       Microchip PIC16C54 to a 24LCXX Serial EE device
0007            ;       and perform a random read operation. This program
0008            ;       will read 8 consecutive addresses in the 'random
0009            ;       read' mode.  This entails setting the address pointer
0010            ;       before doing read command for every address.
0011            ;
0012            ;       Another, more efficient method of reading consecutive
0013            ;       addresses is the 'sequential read' mode.  This involves
0014            ;       sending the control byte and address for the first byte
0015            ;       to read, then continuing to provide clocks for the next
0016            ;       addresses.  The device will automatically increment the
0017            ;       address.  An example of the sequential read mode is
0018            ;       provided in the '2wseqr.asm' file.
0019            ;
0020            ;       Timing is based on using the PIC16C54 in 'XT' mode
0021            ;       using a 4MHz crystal.  Clock speeds to the serial EE
0022            ;       will be approximately 60 kHz for this setup.
0023            ;
0024            ;       As an option, the user may connect a LED to pin 18
0025            ;       on the PIC16C54 (use about a 1K resistor in series) as an
0026            ;       acknowledge fail indicator.  This LED will come on if
0027            ;       the serial EE fails to acknowledge correctly.
0028            ;
0029            ;        PIC16C54 to Serial EE Connections:
0030            ;
0031            ;        PIC16C54        Serial EE
0032            ;       ______    ______
0033            ;       Pin 12 (RB6) ->  SCLK
0034            ;       Pin 13 (RB7) ->  SDATA
0035            ;
0036            ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional)
0037            ;
0038            ;
0039            ;*****************************************************************
0040            ;               Register Definitions
0041            ;*****************************************************************
0042    0003    status  equ     3h      ; status register
0043    0005    port_a  equ     5h      ; port 5 (port a) used for LED display
0044    0006    port_b  equ     6h      ; port 6 (port b) used for data and
0045                                    ; clock lines
0046    000A    eeprom  equ     0ah     ; bit buffer
0047    000B    bycnt   equ     0bh     ; byte counter for read mode
0048    000C    addr    equ     0ch     ; address counter
0049    000D    datai   equ     0dh     ; data input register
0050    000E    datao   equ     0eh     ; data output register
0051    000F    slave   equ     0fh     ; device address 1010xxx0)

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:23 1994  Page 2

Line   PC    Opcode

0052    0010    txbuf   equ     10h     ; transmit buffer
0053    0011    count   equ     11h     ; bit counter
0054    0012    bcount  equ     12h     ; byte counter
0055    0015    loops   equ     15h     ; delay loop counter
0056    0016    loops2  equ     16h     ; delay loop counter
0057
0058            ;*****************************************************************
0059            ;               Bit Definitions
0060            ;*****************************************************************
```


© 1994 Microchip Technology Inc.

```
0061          0007  di      equ     7       ; eeprom input bit
0062          0006  do      equ     6       ; eeprom output bit
0063          0007  sdata   equ     7       ; serial EE data line (port_b,pin 13)
0064          0006  sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0065          0001  ackf    equ     1       ; acknowledge fail LED (port_a)
0066                ;*****************************************************************
0067                ;
0068          0000          org     01ffh   ; set reset vector
0069    01FF  0A5E          goto    PWRUP
0070          0000          org     000h    ;
0071    0000  0A5E          goto    PWRUP
0072                ;
0073                ;*****************************************************************
0074                ;       Start Bit Subroutine
0075                ;       this routine generates a start bit
0076                ;       (Low going data line while clock is high)
0077                ;*****************************************************************
0078                ;
0079                BSTART
0080    0001  05E6            bsf     port_b,sdata    ; make sure data is high
0081    0002  0C3F            movlw   b'00111111'
0082    0003  0006            tris    port_b          ; set data and clock lines for output
0083    0004  04C6            bcf     port_b,sclk     ; make sure clock is low
0084    0005  0000            nop
0085    0006  05C6            bsf     port_b,sclk     ; set clock high
0086    0007  0000            nop
0087    0008  0000            nop
0088    0009  0000            nop
0089    000A  0000            nop
0090    000B  0000            nop
0091    000C  04E6            bcf     port_b,sdata    ; data line goes low during
0092                                                  ; high clock for start bit
0093    000D  0000            nop
0094    000E  0000            nop
0095    000F  0000            nop
0096    0010  0000            nop
0097    0011  0000            nop                     ; timing adjustment
0098    0012  04C6            bcf     port_b,sclk     ; start clock train
0099    0013  0000            nop
0100    0014  0000            nop
0101    0015  0800            retlw   0
0102                          ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:23 1994  Page 3

Line    PC    Opcode

0103                ;*****************************************************************
0104                ;       Stop Bit Subroutine
0105                ;       This routine generates a stop bit
0106                ;       (High going data line while clock is high)
0107                ;*****************************************************************
0108                BSTOP
0109    0016  04E6            bcf     port_b,sdata    ; make sure data line is low
0110    0017  0C3F            movlw   b'00111111'     ;
0111    0018  0006            tris    port_b          ; set data/clock lines as outputs
0112    0019  04E6            bcf     port_b,sdata    ; make sure data line is low
0113    001A  0000            nop
0114    001B  0000            nop
0115    001C  0000            nop
0116    001D  05C6            bsf     port_b,sclk     ; set clock high
0117    001E  0000            nop
0118    001F  0000            nop
0119    0020  0000            nop
0120    0021  05E6            bsf     port_b,sdata    ; data goes high while clock high
0121                                                  ; for stop bit
```

8

© 1994 Microchip Technology Inc.

```
0122   0022  0000           nop
0123   0023  0000           nop
0124   0024  04C6           bcf     port_b,sclk     ; set clock low again
0125   0025  0000           nop
0126   0026  0000           nop
0127   0027  0000           nop
0128   0028  0800           retlw   0
0129                 ;
0130                 ;*****************************************************************
0131                 ;       BITOUT routine takes the bit of data in 'do' and
0132                 ;       transmits it to the serial EE device
0133                 ;*****************************************************************
0134                 BITOUT
0135   0029  0C3F           movlw   b'00111111'     ; set data,clock as outputs
0136   002A  0006           tris    port_b
0137   002B  07CA           btfss   eeprom,do       ; check for state of data bit to xmit
0138   002C  0A2F           goto    bitlow          ; low? go set data line low
0139   002D  05E6           bsf     port_b,sdata    ; high? set data line high
0140   002E  0A30           goto    clkout          ; go toggle the clock
0141
0142   002F  04E6  bitlow    bcf     port_b,sdata    ; output a low bit
0143   0030  05C6  clkout    bsf     port_b,sclk     ; set clock line high
0144   0031  0000           nop
0145   0032  0000           nop
0146   0033  0000           nop
0147   0034  0000           nop
0148   0035  04C6           bcf     port_b,sclk     ; return clock line low
0149   0036  0800           retlw   0
0150                 ;
0151                 ;*****************************************************************
0152                 ;       BITIN routine reads one bit of data from the
0153                 ;       serial EE device and stores it in the bit 'di'
```

16c5x/7x Cross-Assembler V4.12 Released Mon Jun 06 10:48:23 1994 Page 4

```
Line   PC    Opcode

0154                 ;*****************************************************************
0155                 BITIN
0156   0037  05EA           bsf     eeprom,di       ; assume input bit is high
0157   0038  0CBF           movlw   b'10111111'     ; make sdata an input line
0158   0039  0006           tris    port_b
0159   003A  05C6           bsf     port_b,sclk     ; set clock line high
0160   003B  0000           nop                     ; just sit here a sec
0161   003C  0000           nop
0162   003D  0000           nop
0163   003E  0000           nop
0164   003F  0000           nop                     ;
0165   0040  07E6           btfss   port_b,sdata    ; read the data bit
0166   0041  04EA           bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0167   0042  04C6           bcf     port_b,sclk     ; set clock line low
0168                                                ;
0169   0043  0800           retlw   0               ;
0170                ;
0171                ;*****************************************************************
0172                ;       Transmit Data Subroutine
0173                ;       This routine takes the byte of data stored in the
0174                ;       'datao' register and transmits it to the serial EE device.
0175                ;       It will then send 1 more clock to the serial EE for the
0176                ;       acknowledge bit.  If the ack bit from the part was low
0177                ;       then the transmission was successful.  If it is high, then
0178                ;       the device did not send a proper ack bit and the ack
0179                ;       fail LED will be turned on.
0180                ;*****************************************************************
0181                TX
0182   0044  0C08           movlw   .8
0183   0045  0031           movwf   count           ; set the #bits to 8
0184                        ;
0185                TXLP
```


© 1994 Microchip Technology Inc.

```
0186   0046  04CA          bcf     eeprom,do       ; assume bit out is low
0187   0047  06F0          btfsc   txbuf,7         ; is bit out really low?
0188   0048  05CA          bsf     eeprom,do       ; no, set it high
0189   0049  0929          call    BITOUT          ; send the bit to serial EE
0190   004A  0370          rlf     txbuf           ; rotate txbuf left
0191   004B  02F1          decfsz  count           ; 8 bits done?
0192   004C  0A46          goto    TXLP            ; no - go again
0193   004D  0937          call    BITIN           ; read ack bit
0194   004E  06EA          btfsc   eeprom,di       ; check ack bit
0195   004F  0525          bsf     port_a,ackf     ; set acknowledge fail LED if the
0196                                               ; device did not pull data low
0197                       ;
0198   0050  0800          retlw   0
0199                ;
0200                ;*****************************************************************
0201                ;  Receive data routine
0202                ;  This routine reads one byte of data from the part
0203                ;  into the 'datai' register.  It then sends a high
0204                ;  ack bit to indicate that no more data is to be read

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:23 1994  Page 5

Line   PC    Opcode

0205                ;*****************************************************************
0206                RX
0207   0051  006D          clrf    datai           ; clear input buffer
0208   0052  0C08          movlw   .8              ; set # bits to 8
0209   0053  0031          movwf   count
0210   0054  0403          bcf     status,0        ; make sure carry bit is low
0211   0055  036D  RXLP    rlf     datai           ; rotate datai 1 bit left
0212   0056  0937          call    BITIN           ; read a bit
0213   0057  06EA          btfsc   eeprom,di
0214   0058  050D          bsf     datai,0         ; set bit 0 if necessary
0215   0059  02F1          decfsz  count           ; 8 bits done?
0216   005A  0A55          goto    RXLP            ; no, do another
0217   005B  05CA          bsf     eeprom,do       ; set ack bit = 1
0218   005C  0929          call    BITOUT          ; to finish transmission
0219   005D  0800          retlw   0
0220                ;
0221                ;*****************************************************************
022           0000  END
```

8

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002               ;*************************************************************
0003               ;       2-Wire Byte Write Program (123 bytes)
0004               ;
0005               ;       This program demonstrates how to interface a
0006               ;       Microchip PIC16C54 to a 24LCXX Serial EE device
0007               ;       and perform a byte write operation on 8 consecutive
0008               ;       addresses.
0009               ;
0010               ;       This routine waits approximately 10mS for the write
0011               ;       cycle time, which is enough time for any of the
0012               ;       24LCxx devices to complete a write.  A more efficient
0013               ;       method of determining when the write cycle is complete
0014               ;       is called "data polling."  The data polling method is
0015               ;       explained in the program "2wdpoll.asm."  That
0016               ;       particular program uses data polling in a byte write
0017               ;       mode but it can be used in exactly the same way for
0018               ;       a page mode write.
0019               ;
0020               ;       As an option, the user may connect a LED to pin 18
0021               ;       on the PIC16C54 (use about a 1K resistor in series) as an
0022               ;       acknowledge fail indicator.  This LED will come on if
0023               ;       the serial EE fails to acknowledge correctly.
0024               ;
0025               ;       Timing is based on using the PIC16C54 in 'XT' mode
0026               ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0027               ;       will be approximately 60 kHz for this setup.
0028               ;
0029               ;        PIC16C54 to Serial EE Connections:
0030               ;
0031               ;        PIC16C54          Serial EE
0032               ;        --------          ---------
0033               ;       Pin 12 (RB6) -->   SCLK
0034               ;       Pin 13 (RB7) -->   SDATA
0035               ;
0036               ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
0037               ;
0038               ;*************************************************************
0039               ;           Register Definitions
0040               ;*************************************************************
0041         0005  port_a  equ     5h      ; port 5 (port_a) used for LEDs
0042         0006  port_b  equ     6h      ; port 6 (port b) used for data and
0043                                       ; clock lines
0044         000A  eeprom  equ     0ah     ; bit buffer
0045         000B  bycnt   equ     0bh     ; byte counter for read mode
0046         000C  addr    equ     0ch     ; address counter
0047         000D  datai   equ     0dh     ; data input register
0048         000E  datao   equ     0eh     ; data output register
0049         000F  slave   equ     0fh     ; device address
0050                                       ; (1010xxx0)
0051         0010  txbuf   equ     10h     ; transmit buffer


16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 2

Line   PC    Opcode

0052         0011  count   equ     11h     ; bit counter
0053         0012  bcount  equ     12h     ; byte counter
0054         0015  loops   equ     15h     ; delay loop counter
0055         0016  loops2  equ     16h     ; delay loop counter
0056               ;*************************************************************
0057               ;           Bit Definitions
0058               ;*************************************************************
0059         0007  di      equ     7       ; eeprom input bit
```


© 1994 Microchip Technology Inc.

```
0060          0006  do      equ     6       ; eeprom output bit
0061          0007  sdata   equ     7       ; serial EE data line (port_b,pin 13)
0062          0006  sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0063          0001  ackf    equ     1       ; acknowledge fail LED (port_a,pin 18)
0064                ;*************************************************************
0065                ;
0066                ;
0067          0000          org     01ffh   ; set reset vector
0068    01FF  0A5E          goto    PWRUP
0069          0000          org     000h    ;
0070    0000  0A5E          goto    PWRUP
0071                ;
0072                ;*************************************************************
0073                ;       DELAY ROUTINE
0074                ;       This routine takes the value in 'loops'
0075                ;       and multiplies it times 1 millisecond to
0076                ;       determine delay time.
0077                ;*************************************************************
0078                WAIT
0079                ;
0080    0001  0C6E  top       movlw   .110          ; timing adjustment variable
0081    0002  0036            movwf   loops2
0082    0003  0000  top2      nop                   ; sit and wait
0083    0004  0000            nop
0084    0005  0000            nop
0085    0006  0000            nop
0086    0007  0000            nop
0087    0008  0000            nop
0088    0009  02F6            decfsz  loops2        ; inner loops complete?
0089    000A  0A03            goto    top2          ; no, go again
0090                                                ;
0091    000B  02F5            decfsz  loops         ; outer loops complete?
0092    000C  0A01            goto    top           ; no, go again
0093    000D  0800            retlw   0             ; yes, return from sub
0094                ;
0095                ;*************************************************************
0096                ;       Start Bit Subroutine
0097                ;       this routine generates a start bit
0098                ;       (Low going data line while clock is high)
0099                ;*************************************************************
0100                ;
0101                BSTART
0102    000E  05E6            bsf     port_b,sdata  ; make sure data is high


16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 3

Line    PC    Opcode

0103    000F  0C3F            movlw   b'00111111'
0104    0010  0006            tris    port_b        ; set data and clock lines for output
0105    0011  04C6            bcf     port_b,sclk   ; make sure clock is low
0106    0012  0000            nop
0107    0013  05C6            bsf     port_b,sclk   ; set clock high
0108    0014  0000            nop
0109    0015  0000            nop
0110    0016  0000            nop
0111    0017  0000            nop
0112    0018  0000            nop
0113    0019  04E6            bcf     port_b,sdata  ; data line goes low during
0114                                                ; high clock for start bit
0115    001A  0000            nop
0116    001B  0000            nop
0117    001C  0000            nop
0118    001D  0000            nop
0119    001E  0000            nop           ; timing adjustment
0120    001F  04C6            bcf     port_b,sclk   ; start clock train
0121    0020  0000            nop
0122    0021  0000            nop
```


© 1994 Microchip Technology Inc.

```
0123   0022  0800          retlw   0
0124                       ;
0125                       ;  End of Subroutine
0126                       ;*************************************************************
0127                       ;       Stop Bit Subroutine
0128                       ;       This routine generates a stop bit
0129                       ;       (High going data line while clock is high)
0130                       ;*************************************************************
0131                       BSTOP
0132   0023  0C3F          movlw   b'00111111'     ;
0133   0024  0006          tris    port_b          ; set data/clock lines as outputs
0134   0025  04E6          bcf     port_b,sdata    ; make sure data line is low
0135   0026  0000          nop
0136   0027  0000          nop
0137   0028  0000          nop
0138   0029  05C6          bsf     port_b,sclk     ; set clock high
0139   002A  0000          nop
0140   002B  0000          nop
0141   002C  0000          nop
0142   002D  05E6          bsf     port_b,sdata    ; data goes high while clock high
0143                                               ; for stop bit
0144   002E  0000          nop
0145   002F  0000          nop
0146   0030  04C6          bcf     port_b,sclk     ; set clock low again
0147   0031  0000          nop
0148   0032  0000          nop
0149   0033  0000          nop
0150   0034  0800          retlw   0
0151                       ;
0152                       ;          End of Subroutine
0153                       ;*************************************************************


16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 4

Line   PC    Opcode

0154                       ;       BITOUT routine takes one bit of data in 'do' and
0155                       ;       transmits it to the serial EE device
0156                       ;*************************************************************
0157                       BITOUT
0158   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0159   0036  0006          tris    port_b
0160   0037  07CA          btfss   eeprom,do       ; check for state of data bit to xmit
0161   0038  0A3B          goto    bitlow          ;
0162   0039  05E6          bsf     port_b,sdata    ; set data line high
0163   003A  0A3C          goto    clkout          ; go toggle the clock
0164
0165   003B  04E6  bitlow   bcf     port_b,sdata    ; output a low bit
0166   003C  05C6  clkout   bsf     port_b,sclk     ; set clock line high
0167   003D  0000          nop
0168   003E  0000          nop
0169   003F  0000          nop
0170   0040  0000          nop
0171   0041  04C6          bcf     port_b,sclk     ; return clock line low
0172   0042  0800          retlw   0
0173                       ;
0174                       ;                  End of Subroutine
0175                       ;
0176                       ;*************************************************************
0177                       ;       BITIN routine reads one bit of data from the
0178                       ;       serial EE device and stores it in 'di'
0179                       ;*************************************************************
0180                       BITIN
0181   0043  05EA          bsf     eeprom,di       ; assume input bit is high
0182   0044  0CBF          movlw   b'10111111'     ; make sdata an input line
0183   0045  0006          tris    port_b
0184   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
0185   0047  05C6          bsf     port_b,sclk     ; set clock line high
```


© 1994 Microchip Technology Inc.

```
0186   0048  0000            nop                     ; just sit here a sec
0187   0049  0000            nop
0188   004A  0000            nop
0189   004B  0000            nop
0190   004C  0000            nop                     ;
0191   004D  07E6            btfss   port_b,sdata    ; read the data bit
0192   004E  04EA            bcf     eeprom,di       ; input bit was low
0193   004F  04C6            bcf     port_b,sclk     ; set clock line low
0194                                                  ;
0195   0050  0800            retlw   0               ;
0196                 ;
0197                 ;*****************************************************************
0198                 ;       Transmit Data Subroutine
0199                 ;       This routine takes the byte of data stored in the
0200                 ;       'datao' register and transmits it to the serial EE device.
0201                 ;       It will then send 1 more clock to the serial EE for the
0202                 ;       acknowledge bit.  If the ack bit from the part was low
0203                 ;       then the transmission was successful.  If it is high, then
0204                 ;       the device did not send a proper ack bit and the ack


16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 5

Line   PC    Opcode

0205                 ;       fail LED will be turned on.
0206                 ;*****************************************************************
0207                 TX
0208   0051  0C08            movlw   .8
0209   0052  0031            movwf   count           ; set the #bits to 8
0210                         ;
0211                 TXLP
0212   0053  04CA            bcf     eeprom,do       ; assume bit out is low
0213   0054  06F0            btfsc   txbuf,7         ; is bit out really low?
0214   0055  05CA            bsf     eeprom,do       ; otherwise data bit =1
0215   0056  0935            call    BITOUT          ; serial data out
0216   0057  0370            rlf     txbuf           ; rotate txbuf left
0217   0058  02F1            decfsz  count           ; 8 bits done?
0218   0059  0A53            goto    TXLP            ; no - go again
0219   005A  0943            call    BITIN           ; read ack bit
0220   005B  06EA            btfsc   eeprom,di       ; check ack bit
0221   005C  0525            bsf     port_a,ackf     ; set acknowledge fail LED if the
0222                         ;
0223   005D  0800            retlw   0
0224                 ;
0225                 ;*****************************************************************
0226                 ;       Power up routine
0227                 ;       This is the program entry point, which in this case simply
0228                 ;       sets the port_a I/O lines and directs control to the
0229                 ;       byte write routine.
0230                 ;*****************************************************************
0231                 PWRUP
0232   005E  0C00             movlw  b'00000000'
0233   005F  0005             tris   port_a          ; set port A as all output
0234   0060  0065             clrf   port_a          ; all output lines low
0235   0061  0A62             goto   WRBYTE
0236
0237                 ;*****************************************************************
0238                 ;       Byte Write Routine
0239                 ;       This routine writes the data in "datao" to
0240                 ;       8 consecutive bytes in the serial EE device starting
0241                 ;       at address 00.  This routine waits 10mS after every
0242                 ;       byte to give the device time to do the write.  This
0243                 ;       program repeats forever.
0244                 ;*****************************************************************
0245                 ;
0246                 WRBYTE
0247                 ;
0248   0062  0065            clrf    port_a          ; clear all LEDs
```


© 1994 Microchip Technology Inc.

```
0249   0063  0CA0          movlw   b'10100000' ; set slave address and write mode
0250   0064  002F          movwf   slave
0251   0065  0C55          movlw   b'01010101' ; set data to write as 55h
0252   0066  002E          movwf   datao
0253                 ;
0254   0067  0C08          movlw   .8          ; set number of bytes
0255   0068  0032          movwf   bcount      ; to write as to 8


16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:29 1994  Page 6

Line   PC    Opcode

0256   0069  006C          clrf    addr        ; set starting address to 00
0257                                           ;
0258   006A  090E  byte     call    BSTART      ; generate start bit
0259   006B  020F          movf    slave,w     ; move slave address
0260   006C  0030          movwf   txbuf       ; into transmit buffer
0261   006D  0951          call    TX          ; and send it
0262   006E  020C          movf    addr,w      ; move word address
0263   006F  0030          movwf   txbuf       ; into transmit buffer
0264   0070  0951          call    TX          ; and send it
0265   0071  020E          movf    datao,w     ; move data byte
0266   0072  0030          movwf   txbuf       ; to tranmit buffer
0267   0073  0951          call    TX          ; and transmit it
0268   0074  0923          call    BSTOP       ; generate stop bit
0269                                           ;
0270   0075  0C0A          movlw   .10
0271   0076  0035          movwf   loops       ; set delay time to give
0272   0077  0901          call    WAIT        ; 10 ms wait after every byte
0273   0078  02AC          incf    addr        ; add 1 to address counter
0274   0079  02F2          decfsz  bcount      ; all 8 bytes written?
0275   007A  0A6A          goto    byte        ; no, do another byte
0276                                           ;
0277   007B  0A62          goto    wrbyte      ; start over
0278                 ;
0279                 ;
0280         0000  END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002               ;*************************************************************
0003               ;       2-Wire Byte Write With Data Polling Program (129 bytes)
0004               ;
0005               ;       This program demonstrates how to interface a
0006               ;       Microchip PIC16C54 to a 24LCXX Serial EE device.
0007               ;       This program performs a byte write operation on 8
0008               ;       consecutive addresses using the data polling method
0009               ;       to determine when the write cycle is complete.
0010               ;
0011               ;       When writing to a serial E≤ device, there are 2
0012               ;       ways to handle the internal timed write cycle
0013               ;       time of the device.  The simplest method is
0014               ;       simply to wait until the maximum cycle time
0015               ;       is exceeded before attempting another command.
0016               ;       The other, more efficient method is known as "data
0017               ;       polling."  Data polling is done by sending a start
0018               ;       bit and control byte to the part after the write
0019               ;       cycle has been initiated by a stop bit.  If the ack bit
0020               ;       is low, then the device is through writing, otherwise
0021               ;       the the sequence is repeated.  If no low acknowledge
0022               ;       is found within 40 attempts (about 10 milliseconds)
0023               ;       then the routine times out and sets the timeout
0024               ;       LED (pin 1) high.
0025               ;
0026               ;       As an option, the user can connect a LED to pin 1
0027               ;       on the PIC16C54 (use about a 1K resistor in series) as a
0028               ;       timeout indicator.  This LED will come on if the
0029               ;       data polling is unsuccessful and the device being
0030               ;       programmed does not respond.  This can be tested by
0031               ;       simply running the program with no part in the socket.
0032               ;
0033               ;       Timing is based on using the PIC16C54 in 'XT' mode
0034               ;       using a 4MHz crystal.  Clock speeds to the serial EE
0035               ;       will be approximately 40 kHz for this setup.
0036               ;
0037               ;        PIC16C54 to Serial EE Connections:
0038               ;
0039               ;        PIC16C54          Serial EE
0040               ;        --------          ---------
0041               ;       Pin 12 (RB6) ->  SCLK
0042               ;       Pin 13 (RB7) ->  SDATA
0043               ;
0044               ;       Pin 1  (RA2) ->  Write cycle timeout fail LED (optional)
0045               ;
0046               ;*************************************************************
0047               ;          Register Definitions
0048               ;*************************************************************
0049          0005 port_a  equ     5h      ; port 5 (port_a) used LEDs
0050          0006 port_b  equ     6h      ; port 6 (port_b) used for data and
0051                                       ; clock lines

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 2

Line   PC    Opcode

0052          000A eeprom  equ     0ah     ; bit buffer
0053          000B bycnt   equ     0bh     ; byte counter for read mode
0054          000C addr    equ     0ch     ; address counter
0055          000D datai   equ     0dh     ; data input register
0056          000E datao   equ     0eh     ; data output register
0057          000F slave   equ     0fh     ; device address (1010xxx0)
0058          0010 txbuf   equ     10h     ; transmit buffer
0059          0011 count   equ     11h     ; bit counter
0060          0012 bcount  equ     12h     ; byte counter
```

8

© 1994 Microchip Technology Inc.

```
0061          0015  loops   equ    15h      ; delay loop counter
0062          0016  loops2  equ    16h      ; delay loop counter
0063          0017  pollcnt equ    17h      ; data polling counter
0064                ;*************************************************************
0065                ;           Bit Definitions
0066                ;*************************************************************
0067          0007  di      equ    7        ; eeprom input bit
0068          0006  do      equ    6        ; eeprom output bit
0069          0007  sdata   equ    7        ; serial EE data line (port_b,pin 13)
0070          0006  sclk    equ    6        ; serial EE clock line (port_b,pin 12)
0071          0002  timeout equ    2        ; write cycle timeout fail LED, port_a (pin 1)
0072          0001  ackf    equ    1        ; acknowledge fail LED, port_a (pin 18)
0073                ;*************************************************************
0074                ;
0075          0000          org    01ffh    ; set reset vector
0076    01FF  0A5C          goto   PWRUP
0077          0000          org    000h     ;
0078    0000  0A5C          goto   PWRUP
0079                ;
0080                ;*************************************************************
0081                ;       DELAY ROUTINE
0082                ;       This routine takes the value in 'loops'
0083                ;       and multiplies it times 1 millisecond to
0084                ;       determine delay time.
0085                ;*************************************************************
0086                WAIT
0087                ;
0088    0001  0C6E  top     movlw   .110            ; timing adjustment variable
0089    0002  0036          movwf   loops2
0090    0003  0000  top2    nop                     ; sit and wait
0091    0004  0000          nop
0092    0005  0000          nop
0093    0006  0000          nop
0094    0007  0000          nop
0095    0008  0000          nop
0096    0009  02F6          decfsz  loops2          ; inner loops complete?
0097    000A  0A03          goto    top2            ; no, go again
0098                                                ;
0099    000B  02F5          decfsz  loops           ; outer loops complete?
0100    000C  0A01          goto    top             ; no, go again
0101    000D  0800          retlw   0               ; yes, return from sub
0102                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 3

Line    PC    Opcode

0103                ;*************************************************************
0104                ;       Start Bit Subroutine
0105                ;       this routine generates a start bit
0106                ;       (Low going data line while clock is high)
0107                ;*************************************************************
0108                ;
0109                BSTART
0110    000E  05E6          bsf     port_b,sdata    ; make sure data is high
0111    000F  0C3F          movlw   b'00111111'
0112    0010  0006          tris    port_b          ; set data and clock lines for output
0113    0011  04C6          bcf     port_b,sclk     ; make sure clock is low
0114    0012  0000          nop
0115    0013  05C6          bsf     port_b,sclk     ; set clock high
0116    0014  0000          nop
0117    0015  0000          nop
0118    0016  0000          nop
0119    0017  0000          nop
0120    0018  0000          nop
0121    0019  04E6          bcf     port_b,sdata    ; data line goes low during
0122                                                ; high clock for start bit
0123    001A  0000          nop
0124    001B  0000          nop
```


© 1994 Microchip Technology Inc.

```
0125   001C  0000           nop
0126   001D  0000           nop
0127   001E  0000           nop                    ; timing adjustment
0128   001F  04C6           bcf     port_b,sclk    ; start clock train
0129   0020  0000           nop
0130   0021  0000           nop
0131   0022  0800           retlw   0
0132                        ;
0133                 ;*************************************************************
0134                 ;       Stop Bit Subroutine
0135                 ;       This routine generates a stop bit
0136                 ;       (High going data line while clock is high)
0137                 ;*************************************************************
0138                 BSTOP
0139   0023  0C3F           movlw   b'00111111'    ;
0140   0024  0006           tris    port_b         ; set data/clock lines as outputs
0141   0025  04E6           bcf     port_b,sdata   ; make sure data line is low
0142   0026  0000           nop
0143   0027  0000           nop
0144   0028  0000           nop
0145   0029  05C6           bsf     port_b,sclk    ; set clock high
0146   002A  0000           nop
0147   002B  0000           nop
0148   002C  0000           nop
0149   002D  05E6           bsf     port_b,sdata   ; data goes high while clock high
0150                                               ; for stop bit
0151   002E  0000           nop
0152   002F  0000           nop
0153   0030  04C6           bcf     port_b,sclk    ; set clock low again

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 4

Line   PC    Opcode

0154   0031  0000           nop
0155   0032  0000           nop
0156   0033  0000           nop
0157   0034  0800           retlw   0
0158                 ;
0159                 ;*************************************************************
0160                 ;       BITOUT routine takes one bit of data in 'do' and
0161                 ;       transmits it to the serial EE device
0162                 ;*************************************************************
0163                 BITOUT
0164   0035  0C3F           movlw   b'00111111'    ; set data,clock as outputs
0165   0036  0006           tris    port_b
0166   0037  07CA           btfss   eeprom,do      ; check for state of data bit to xmit
0167   0038  0A3B           goto    bitlow         ;
0168   0039  05E6           bsf     port_b,sdata   ; set data line high
0169   003A  0A3C           goto    clkout         ; go toggle the clock
0170                                               ;
0171   003B  04E6  bitlow   bcf     port_b,sdata   ; output a low bit
0172   003C  05C6  clkout   bsf     port_b,sclk    ; set clock line high
0173   003D  0000           nop
0174   003E  0000           nop
0175   003F  0000           nop
0176   0040  0000           nop
0177   0041  04C6           bcf     port_b,sclk    ; return clock line low
0178   0042  0800           retlw   0
0179                 ;
0180                 ;                  End of Subroutine
0181                 ;
0182                 ;*************************************************************
0183                 ;       BITIN routine reads one bit of data from the
0184                 ;       serial EE device and stores it in 'di'
0185                 ;*************************************************************
0186                 BITIN
0187   0043  05EA           bsf     eeprom,di      ; assume input bit is high
0188   0044  0CBF           movlw   b'10111111'    ; make sdata an input line
```


© 1994 Microchip Technology Inc.

```
0189   0045  0006          tris    port_b
0190   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
0191   0047  05C6          bsf     port_b,sclk     ; set clock line high
0192   0048  0000          nop                     ; just sit here a sec
0193   0049  0000          nop
0194   004A  0000          nop
0195   004B  0000          nop
0196   004C  0000          nop                     ;
0197   004D  07E6          btfss   port_b,sdata    ; read the data bit
0198   004E  04EA          bcf     eeprom,di       ; input bit was low
0199   004F  04C6          bcf     port_b,sclk     ; set clock line low
0200                                               ;
0201   0050  0800          retlw   0               ;
0202                ;
0203                ;*****************************************************************
0204                ;       Transmit Data Subroutine

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 5

Line   PC    Opcode

0205                ;       This routine takes the byte of data stored in the
0206                ;       'datao' register and transmits it to the serial EE device.
0207                ;       It will then send 1 more clock to the serial EE for the
0208                ;       acknowledge bit.  If the ack bit from the part was low
0209                ;       then the transmission was successful.  If it is high, then
0210                ;       the device did not send a proper ack bit and the ack
0211                ;       fail LED will be turned on.
0212                ;*****************************************************************
0213                TX
0214   0051  0C08          movlw   .8
0215   0052  0031          movwf   count           ; set the #bits to 8
0216                       ;
0217                TXLP
0218   0053  04CA          bcf     eeprom,do       ; assume bit out is low
0219   0054  06F0          btfsc   txbuf,7         ; is bit out really low?
0220   0055  05CA          bsf     eeprom,do       ; otherwise data bit =1
0221   0056  0935          call    BITOUT          ; serial data out
0222   0057  0370          rlf     txbuf           ; rotate txbuf left
0223   0058  02F1          decfsz  count           ; 8 bits done?
0224   0059  0A53          goto    TXLP            ; no - go again
0225   005A  0943          call    BITIN           ; read ack bit
0226                       ;
0227   005B  0800          retlw   0
0228                ;                              ;
0229                ;*****************************************************************
0230                ;       Power up routine
0231                ;       This is the program entry point, which in this case simply
0232                ;       sets the port_a I/O lines and directs control to the
0233                ;       write routine.
0234                ;*****************************************************************
0235                PWRUP
0236   005C  0C00          movlw   b'00000000'
0237   005D  0005          tris    port_a          ; set port A as all output
0238   005E  0065          clrf    port_a          ; all output lines low
0239   005F  0A60          goto    WRBYTE
0240
0241                ;*****************************************************************
0242                ;       Byte Write Routine with data polling technique
0243                ;
0244                ;       This routine writes the data in "datao" to
0245                ;       8 consecutive bytes in the serial EE device starting
0246                ;       at address 00. To determine when the write cycle time
0247                ;       of the device, the 'data polling' method is used.  This
0248                ;       involves sending a start bit and control byte to the part
0249                ;       and checking for an acknowledge.  If the ack bit is low, then
0250                ;       the device is through writing, otherwise the the sequence
0251                ;       is repeated.  If no low acknowledge is found within 40
0252                ;       attempts (about 10 milliseconds) then the routine times
```


© 1994 Microchip Technology Inc.

```
0253                ;       out and sets the timeout LED (pin 18) high.  This program
0254                ;       will repeat forever.
0255                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:50 1994  Page 6

Line   PC    Opcode

0256                ;*****************************************************************
0257                ;
0258                WRBYTE
0259                ;
0260   0060  0065           clrf    port_a          ; all LEDs off
0261   0061  0CA0           movlw   b'10100000'     ; set slave address and write mode
0262   0062  002F           movwf   slave
0263   0063  0CAA           movlw   b'10101010'     ; set data to write as AAh
0264   0064  002E           movwf   datao
0265   0065  0C08           movlw   .8              ; set number of bytes
0266   0066  0032           movwf   bcount          ; to write as to 8
0267   0067  006C           clrf    addr            ; set starting address to 00
0268                                                ;
0269   0068  090E  byte      call    BSTART          ; generate start bit
0270   0069  020F           movf    slave,w         ; move slave address
0271   006A  0030           movwf   txbuf           ; into transmit buffer
0272   006B  0951           call    TX              ; and send it
0273   006C  020C           movf    addr,w          ; move word address
0274   006D  0030           movwf   txbuf           ; into transmit buffer
0275   006E  0951           call    TX              ; and send it
0276   006F  020E           movf    datao,w         ; move data byte
0277   0070  0030           movwf   txbuf           ; to tranmit buffer
0278   0071  0951           call    TX              ; and transmit it
0279   0072  0923           call    BSTOP           ; generate stop bit
0280                                                ;
0281                                                ; now start polling for a low ack bit
0282                                                ;
0283   0073  0C28           movlw   .40             ;
0284   0074  0037           movwf   pollcnt         ; set max number of times to poll as 40
0285   0075  090E  poll      call    BSTART          ; generate start bit
0286   0076  0CA0           movlw   b'10100000'     ; move slave address (write mode)
0287   0077  0030           movwf   txbuf           ; into transmit buffer
0288   0078  0951           call    TX              ; and send it
0289   0079  07EA           btfss   eeprom,di       ; was the ack bit low?
0290   007A  0A7E           goto    exitpoll        ; yes, do another byte
0291   007B  02F7           decfsz  pollcnt         ; is poll counter down to zero?
0292   007C  0A75           goto    poll            ; no, poll again.  Other wise the part is
0293   007D  0545           bsf     port_a,timeout  ; not responding in time so set timeout
0294                                                ; LED and continue on
0295                                                ;
0296   007E  02AC  exitpoll  incf    addr            ; add 1 to address counter
0297   007F  02F2           decfsz  bcount          ; all 8 bytes written?
0298   0080  0A68           goto    byte            ; no, do another byte
0299   0081  0A60           goto    WRBYTE          ; yes, start over
0300                ;
0301                ;
0302         0000  END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       2-Wire Page Write Program (122 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 24LCXX Serial EE device
0007                ;       and perform a page write operation on 8 consecutive
0008                ;       addresses.
0009                ;
0010                ;       All of Microchip's 2-wire serial E≤ devices have a
0011                ;       buffer or 'page' that can be used as a more efficient
0012                ;       method of writing to consecutive addresses.  Some
0013                ;       devices have page lengths of 2 bytes, others have
0014                ;       page lengths of 4,8,16 or 64 bytes.  Please consult
0015                ;       the databook for the page length of the device you
0016                ;       are using.  THIS ROUTINE IS WRITTEN FOR A DEVICE
0017                ;       WITH A PAGE LENGTH OF 8 OR MORE BYTES.
0018                ;
0019                ;       When using page mode, the control byte and address
0020                ;       are sent for the first address only.  After the data
0021                ;       byte for the first address is sent, the data for the
0022                ;       next consecutive address is clocked in.  This is
0023                ;       repeated as many times as needed (as long as the page
0024                ;       length is not exceeded) and then a stop bit is sent.
0025                ;       The device will still acknowledge between every byte of
0026                ;       data.  After the stop bit is sent, the part will
0027                ;       initiate the self timed write cycle.  For all of
0028                ;       Microchip's 24LCxx devices, the cycle time for
0029                ;       a byte write and a page write is the same.  Therefore,
0030                ;       writing 8 bytes in byte mode with a typical 5ms write
0031                ;       cycle consumes 40ms of wait time, while the same data
0032                ;       written in page mode would consume only 5ms of wait time.
0033                ;
0034                ;       This routine waits approximately 10mS for the write
0035                ;       cycle time, which is enough time for any of the
0036                ;       24LCxx devices to complete a write.  A more efficient
0037                ;       method of determining when the write cycle is complete
0038                ;       is called "data polling."  The data polling method is
0039                ;       explained in the program "2wdpoll.asm."  That
0040                ;       particular program uses data polling in a byte write
0041                ;       mode but it can be used in exactly the same way for
0042                ;       a page mode write.
0043                ;
0044                ;       As an option, the user can connect a LED to pin 18
0045                ;       on the PIC16C54 (use about a 1K resistor in series) as a
0046                ;       acknowledge fail indicator.  This LED will come on
0047                ;       if the device being programmed does not send a low
0048                ;       acknowledge bit at the proper times.  This can be
0049                ;       tested by simply running the program with no part
0050                ;       in the socket.
0051                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 2

Line   PC    Opcode

0052                ;       Timing is based on using the PIC16C5x in 'XT' mode
```


© 1994 Microchip Technology Inc.

```
0053                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0054                ;       will be approximately 40 kHz for this setup.
0055                ;
0056                ;
0057                ;        PIC16C54 to Serial EE Connections:
0058                ;
0059                ;          PIC16C54      Serial EE
0060                ;          --------      ---------
0061                ;          Pin 12 (RB6) ->  SCLK
0062                ;          Pin 13 (RB7) ->  SDATA
0063                ;
0064                ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional)
0065 ************** ;
0066                ;
0067                ;*************************************************************
0068                ;                   Register Definitions
0069 ************** ;*************************************************************
0070         0005   port_a  equ     5h      ; port 5 (port a) used for LED outputs
0071         0006   port_b  equ     6h      ; port 6 (port b) used for data and
0072                                        ; clock lines
0073         000A   eeprom  equ     0ah     ; bit buffer
0074         000C   addr    equ     0ch     ; address counter
0075         000E   datao   equ     0eh     ; data output register
0076         000F   slave   equ     0fh     ; device address (1010xxx0)
0077         0010   txbuf   equ     10h     ; transmit buffer
0078         0011   count   equ     11h     ; bit counter
0079         0012   bcount  equ     12h     ; byte counter
0080         0015   loops   equ     15h     ; delay loop counter
0081         0016   loops2  equ     16h     ; delay loop counter
0082                ;*************************************************************
0083                ;              Bit Definitions
0084                ;*************************************************************
0085         0007   di      equ     7       ; eeprom input bit
0086         0006   do      equ     6       ; eeprom output bit
0087         0007   sdata   equ     7       ; serial EE data line (port_b,pin 13)
0088         0006   sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0089         0001   ackf    equ     1       ; acknowledge fail LED, port_a (pin 18)
0090                ;*************************************************************
0091                ;
0092                ;
0093         0000           org     01ffh   ; set reset vector
0094   01FF  0A5E           goto    PWRUP
0095         0000           org     000h    ;
0096   0000  0A5E           goto    PWRUP
0097 ************** ;
0098                ;*************************************************************
0099                ;       DELAY ROUTINE
0100                ;       This routine takes the value in 'loops'
0101                ;       and multiplies it times 1 millisecond to
0102                ;       determine delay time.

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 3

Line   PC     Opcode

0103                ;*************************************************************
0104                WAIT
0105                ;
0106   0001   0C6E  top     movlw   .110            ; timing adjustment variable
0107   0002   0036          movwf   loops2
0108   0003   0000  top2    nop                     ; sit and wait
```

```
0109   0004  0000            nop
0110   0005  0000            nop
0111   0006  0000            nop
0112   0007  0000            nop
0113   0008  0000            nop
0114   0009  02F6            decfsz  loops2          ; inner loops complete?
0115   000A  0A03            goto    top2            ; no, go again
0116                                                 ;
0117   000B  02F5            decfsz  loops           ; outer loops complete?
0118   000C  0A01            goto    top             ; no, go again
0119   000D  0800            retlw   0               ; yes, return from sub
0120                 ;
0121                 ;*************************************************************
0122                 ;       Start Bit Subroutine
0123                 ;       this routine generates a start bit
0124                 ;       (Low going data line while clock is high)
0125                 ;*************************************************************
0126                 ;
0127                 BSTART
0128   000E  05E6            bsf     port_b,sdata    ; make sure data is high
0129   000F  0C3F            movlw   b'00111111'
0130   0010  0006            tris    port_b          ; set data and clock lines for
output
0131   0011  04C6            bcf     port_b,sclk     ; make sure clock is low
0132   0012  0000            nop
0133   0013  05C6            bsf     port_b,sclk     ; set clock high
0134   0014  0000            nop
0135   0015  0000            nop
0136   0016  0000            nop
0137   0017  0000            nop
0138   0018  0000            nop
0139   0019  04E6            bcf     port_b,sdata    ; data line goes low during
0140                                                 ; high clock for start bit
0141   001A  0000            nop
0142   001B  0000            nop
0143   001C  0000            nop
0144   001D  0000            nop
0145   001E  0000            nop                     ; timing adjustment
0146   001F  04C6            bcf     port_b,sclk     ; start clock train
0147   0020  0000            nop
0148   0021  0000            nop
0149   0022  0800            retlw   0
0150                         ;
0151                         ;    End of Subroutine
0152                 ;*************************************************************
0153                 ;       Stop Bit Subroutine

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 4

Line   PC    Opcode

0154                 ;       This routine generates a stop bit
0155                 ;       (High going data line while clock is high)
0156                 ;*************************************************************
0157                 BSTOP
0158   0023  0C3F            movlw   b'00111111'     ;
0159   0024  0006            tris    port_b          ; set data/clock lines as outputs
0160   0025  04E6            bcf     port_b,sdata    ; make sure data line is low
0161   0026  0000            nop
0162   0027  0000            nop
0163   0028  0000            nop
```


© 1994 Microchip Technology Inc.

```
0164   0029  05C6          bsf     port_b,sclk     ; set clock high
0165   002A  0000          nop
0166   002B  0000          nop
0167   002C  0000          nop
0168   002D  05E6          bsf     port_b,sdata    ; data goes high while clock high
0169                                               ; for stop bit
0170   002E  0000          nop
0171   002F  0000          nop
0172   0030  04C6          bcf     port_b,sclk     ; set clock low again
0173   0031  0000          nop
0174   0032  0000          nop
0175   0033  0000          nop
0176   0034  0800          retlw   0
0177                ;
0178                ;               End of Subroutine
0179                ;*************************************************************
0180                ;       BITOUT routine
0181                ; This routine takes one bit of data in 'do' and
0182                ;       transmits it to the serial EE device
0183                ;*************************************************************
0184                BITOUT
0185   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0186   0036  0006          tris    port_b
0187   0037  07CA          btfss   eeprom,do       ; check for state of data bit to
xmit
0188   0038  0A3B          goto    bitlow          ;
0189   0039  05E6          bsf     port_b,sdata    ; set data line high
0190   003A  0A3C          goto    clkout          ; go toggle the clock
0191
0192   003B  04E6  bitlow  bcf     port_b,sdata    ; output a low bit
0193   003C  05C6  clkout  bsf     port_b,sclk     ; set clock line high
0194   003D  0000          nop
0195   003E  0000          nop
0196   003F  0000          nop
0197   0040  0000          nop
0198   0041  04C6          bcf     port_b,sclk     ; return clock line low
0199   0042  0800          retlw   0
0200                ;
0201                ;               End of Subroutine
0202                ;
0203                ;*************************************************************
0204                ;       BITIN routine reads one bit of data from the
```



```
Line   PC    Opcode

0205            ; serial EE device and stores it in 'di'
0206            ;*************************************************************
0207                BITIN
0208   0043  05EA          bsf     eeprom,di       ; assume input bit is high
0209   0044  0CBF          movlw   b'10111111'     ; make sdata an input line
0210   0045  0006          tris    port_b
0211   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
0212   0047  05C6          bsf     port_b,sclk     ; set clock line high
0213   0048  0000          nop                     ; just sit here a sec
0214   0049  0000          nop
0215   004A  0000          nop
0216   004B  0000          nop
0217   004C  0000          nop                     ;
0218   004D  07E6          btfss   port_b,sdata    ; read the data bit
```

8

© 1994 Microchip Technology Inc.

```
0219   004E  04EA            bcf     eeprom,di       ; input bit was low
0220   004F  04C6            bcf     port_b,sclk     ; set clock line low
0221                                                  ;
0222   0050  0800            retlw   0               ;
0223                 ;
0224                 ;*****************************************************************
0225                 ;       Transmit Data Subroutine
0226                 ;       This routine takes the byte of data stored in the
0227                 ;       'datao' register and transmits it to the serial EE device.
0228                 ;       It will then send 1 more clock to the serial EE for the
0229                 ;       acknowledge bit.  If the ack bit from the part was low
0230                 ;       then the transmission was successful.  If it is high, then
0231                 ;       the device did not send a proper ack bit and the ack
0232                 ;       fail LED will be turned on.
0233                 ;*****************************************************************
0234                 TX
0235   0051  0C08            movlw   .8
0236   0052  0031            movwf   count           ; set the #bits to 8
0237                         ;
0238                 TXLP
0239   0053  04CA            bcf     eeprom,do       ; assume bit out is low
0240   0054  06F0            btfsc   txbuf,7         ; is bit out really low?
0241   0055  05CA            bsf     eeprom,do       ; otherwise data bit =1
0242   0056  0935            call    BITOUT          ; serial data out
0243   0057  0370            rlf     txbuf           ; rotate txbuf left
0244   0058  02F1            decfsz  count           ; 8 bits done?
0245   0059  0A53            goto    TXLP            ; no - go again
0246   005A  0943            call    BITIN           ; read ack bit
0247   005B  06EA            btfsc   eeprom,di       ; check ack bit
0248   005C  0525            bsf     port_a,ackf     ; set acknowledge fail LED if the
0249                                                  ; device did not pull data low
0250                         ;
0251   005D  0800            retlw   0
0252                 ;
0253                 ;*****************************************************************
0254                 ;       Power up routine
0255                 ;       This is the program entry point, which in this case simply

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 6

Line   PC    Opcode

0256                 ;       sets the port_a I/O lines and directs control to the Page
0257                 ;       write routine.
0258                 ;*****************************************************************
0259                 PWRUP
0260   005E  0C00              movlw   b'00000000'
0261   005F  0005              tris    port_a          ; set port A as all output
0262   0060  0065              clrf    port_a          ; all output lines low
0263   0061  0A62              goto    WRPAGE
0264
0265                 ;*****************************************************************
0266                 ;       Page Write Routine
0267                 ;
0268                 ;       This routine uses page mode to write the data in "datao" to
0269                 ;       8 consecutive bytes in the serial EE device starting
0270                 ;       at address 00. This routine waits 10mS after every
0271                 ;       page to give the device time to do the write.  This
0272                 ;       routine executes forever
0273                 ;*****************************************************************
0274                 ;
```


© 1994 Microchip Technology Inc.

```
0275                    WRPAGE
0276                    ;
0277    0062    0065            clrf    port_a          ; clear all LEDs
0278    0063    0CA0            movlw   b'10100000'     ; set slave address and write mode
0279    0064    002F            movwf   slave
0280    0065    0C55            movlw   b'01010101'     ; set data to write as 55h
0281    0066    002E            movwf   datao
0282                                                    ;
0283    0067    0C08            movlw   .8              ; set number of bytes
0284    0068    0032            movwf   bcount          ; to write as to 8
0285    0069    006C            clrf    addr            ; set starting address to 00
0286                                                    ;
0287    006A    090E            call    BSTART          ; generate start bit
0288    006B    020F            movf    slave,w         ; move slave address
0289    006C    0030            movwf   txbuf           ; into transmit buffer
0290    006D    0951            call    TX              ; and send it
0291
0292    006E    020C            movf    addr,w          ; move word address
0293    006F    0030            movwf   txbuf           ; into transmit buffer
0294    0070    0951            call    TX              ; and send it
0295
0296    0071    020E    byte    movf    datao,w         ; move data byte
0297    0072    0030            movwf   txbuf           ; to tranmit buffer
0298    0073    0951            call    TX              ; and transmit it
0299    0074    02F2            decfsz  bcount          ; all 8 bytes written?
0300    0075    0A71            goto    byte            ; no, do another
0301    0076    0923            call    BSTOP           ; yes,generate stop bit
0302                                                    ;
0303    0077    0C0A            movlw   .10
0304    0078    0035            movwf   loops           ; set delay time to give
0305    0079    0901            call    WAIT            ; 10 ms wait after every byte
0306                                                    ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:36 1994  Page 7

Line    PC      Opcode

0307    007A    0A62            goto    wrpage          ; start over
0308                    ;
0309                    ;
0310            0000    END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       2-Wire Sequential Read Program (120 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 24LCXX Serial EE device
0007                ;       and perform a sequential read operation.  A sequential
0008                ;       read involves setting the address pointer once and then
0009                ;       using the auto-increment ability of the part to read
0010                ;       consecutive addresses by simply providing more clocks.
0011                ;
0012                ;       Timing is based on using the PIC16C54 in 'XT' mode
0013                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0014                ;       will be approximately 60 kHz for this setup.
0015                ;
0016                ;       As an option, the user may connect a LED to pin 18
0017                ;       on the PIC16C54 (use about a 1K resistor in series) as an
0018                ;       acknowledge fail indicator.  This LED will come on if
0019                ;       the serial EE fails to acknowledge correctly.
0020                ;
0021                ;        PIC16C54 to Serial EE Connections:
0022                ;
0023                ;          PIC16C54          Serial EE
0024                ;          --------          ---------
0025                ;          Pin 12 (RB6) ->   SCLK
0026                ;          Pin 13 (RB7) ->   SDATA
0027                ;
0028                ;          PIN 18 (RA1) ->   Acknowledge fail LED  (Optional)
0029                ;
0030                ;*************************************************************
0031                ;                  Register Definitions
0032                ;*************************************************************
0033          0003  status  equ     3h      ; status register
0034          0005  port_a  equ     5h      ; port 5 (port_a) used for LEDs
0035          0006  port_b  equ     6h      ; port 6 (port b) used for data and
0036                                        ; clock lines
0037          000A  eeprom  equ     0ah     ; bit buffer
0038          000B  bycnt   equ     0bh     ; byte counter for read mode
0039          000C  addr    equ     0ch     ; address counter
0040          000D  datai   equ     0dh     ; data input register
0041          000E  datao   equ     0eh     ; data output register
0042          000F  slave   equ     0fh     ; device address (1010xxx0)
0043          0010  txbuf   equ     10h     ; transmit buffer
0044          0011  count   equ     11h     ; bit counter
0045          0012  bcount  equ     12h     ; byte counter
0046          0015  loops   equ     15h     ; delay loop counter
0047          0016  loops2  equ     16h     ; delay loop counter
0048                ;*************************************************************
0049                ;                  Bit Definitions
0050                ;*************************************************************
0051          0007  di      equ     7       ; eeprom input bit

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 2

Line   PC    Opcode

0052          0006  do      equ     6       ; eeprom output bit
0053          0007  sdata   equ     7       ; serial EE data line (port_b,pin 13)
0054          0006  sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0055          0001  ackf    equ     1       ; acknowledge fail LED, port_a
0056                ;*************************************************************
0057                ;
0058                ;
0059          0000          org     01ffh   ; set reset vector
0060   01FF   0A5B          goto    PWRUP
```


© 1994 Microchip Technology Inc.

```
0061        0000            org     000h    ;
0062  0000   0A5B            goto    PWRUP
0063                  ;
0064                  ;*************************************************************
0065                  ;       Start Bit Subroutine
0066                  ;       this routine generates a start bit
0067                  ;       (Low going data line while clock is high)
0068                  ;*************************************************************
0069                  ;
0070                  BSTART
0071  0001   05E6            bsf     port_b,sdata    ; make sure data is high
0072  0002   0C3F            movlw   b'00111111'
0073  0003   0006            tris    port_b          ; set data and clock lines for output
0074  0004   04C6            bcf     port_b,sclk     ; make sure clock is low
0075  0005   0000            nop
0076  0006   05C6            bsf     port_b,sclk     ; set clock high
0077  0007   0000            nop
0078  0008   0000            nop
0079  0009   0000            nop
0080  000A   0000            nop
0081  000B   0000            nop
0082  000C   04E6            bcf     port_b,sdata    ; data line goes low during
0083                                                  ; high clock for start bit
0084  000D   0000            nop
0085  000E   0000            nop
0086  000F   0000            nop
0087  0010   0000            nop
0088  0011   0000            nop                     ; timing adjustment
0089  0012   04C6            bcf     port_b,sclk     ; start clock train
0090  0013   0000            nop
0091  0014   0000            nop
0092  0015   0800            retlw   0
0093                          ;
0094                          ;    End of Subroutine
0095                  ;*************************************************************
0096                  ;       Stop Bit Subroutine
0097                  ;       This routine generates a stop bit
0098                  ;       (High going data line while clock is high)
0099                  ;*************************************************************
0100                  BSTOP
0101  0016   0C3F            movlw   b'00111111'     ;
0102  0017   0006            tris    port_b          ; set data/clock lines as outputs

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 3

Line    PC    Opcode

0103  0018   04E6            bcf     port_b,sdata    ; make sure data line is low
0104  0019   0000            nop
0105  001A   0000            nop
0106  001B   0000            nop
0107  001C   05C6            bsf     port_b,sclk     ; set clock high
0108  001D   0000            nop
0109  001E   0000            nop
0110  001F   0000            nop
0111  0020   05E6            bsf     port_b,sdata    ; data goes high while clock high
0112                                                  ; for stop bit
0113  0021   0000            nop
0114  0022   0000            nop
0115  0023   04C6            bcf     port_b,sclk     ; set clock low again
0116  0024   0000            nop
0117  0025   0000            nop
0118  0026   0000            nop
0119  0027   0800            retlw   0
0120                  ;
0121                  ;          End of Subroutine
0122                  ;*************************************************************
0123                  ;       BITOUT routine takes one bit of data in 'do' and
0124                  ;       transmits it to the serial EE device
```

8

© 1994 Microchip Technology Inc.

```
0125                    ;*****************************************************************
0126                    BITOUT
0127    0028    0C3F            movlw   b'00111111'     ; set data,clock as outputs
0128    0029    0006            tris    port_b
0129    002A    07CA            btfss   eeprom,do       ; check for state of data bit to xmit
0130    002B    0A2E            goto    bitlow          ;
0131    002C    05E6            bsf     port_b,sdata    ; set data line high
0132    002D    0A2F            goto    clkout          ; go toggle the clock
0133
0134    002E    04E6    bitlow  bcf     port_b,sdata    ; output a low bit
0135    002F    05C6    clkout  bsf     port_b,sclk     ; set clock line high
0136    0030    0000            nop
0137    0031    0000            nop
0138    0032    0000            nop
0139    0033    0000            nop
0140    0034    04C6            bcf     port_b,sclk     ; return clock line low
0141    0035    0800            retlw   0
0142                    ;
0143                    ;                       End of Subroutine
0144                    ;
0145                    ;*****************************************************************
0146                    ;       BITIN routine reads one bit of data from the
0147                    ;       serial EE device and stores it in 'di'
0148                    ;*****************************************************************
0149                    BITIN
0150    0036    05EA            bsf     eeprom,di       ; assume input bit is high
0151    0037    0CBF            movlw   b'10111111'     ; make sdata an input line
0152    0038    0006            tris    port_b
0153    0039    05C6            bsf     port_b,sclk     ; set clock line high

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 4

Line    PC      Opcode

0154    003A    0000            nop                     ; just sit here a sec
0155    003B    0000            nop
0156    003C    0000            nop
0157    003D    0000            nop
0158    003E    0000            nop                     ;
0159    003F    07E6            btfss   port_b,sdata    ; read the data bit
0160    0040    04EA            bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0161    0041    04C6            bcf     port_b,sclk     ; set clock line low
0162                                                    ;
0163    0042    0800            retlw   0
0164                    ;
0165                    ;*****************************************************************
0166                    ;       Transmit Data Subroutine
0167                    ;       This routine takes the byte of data stored in the
0168                    ;       'datao' register and transmits it to the serial EE device.
0169                    ;       It will then send 1 more clock to the serial EE for the
0170                    ;       acknowledge bit.  If the ack bit from the part was low
0171                    ;       then the transmission was successful.  If it is high, then
0172                    ;       the device did not send a proper ack bit and the ack
0173                    ;       fail LED will be turned on.
0174                    ;*****************************************************************
0175                    TX
0176    0043    0C08            movlw   .8
0177    0044    0031            movwf   count           ; set the #bits to 8
0178                            ;
0179                    TXLP
0180    0045    04CA            bcf     eeprom,do       ; assume bit out is low
0181    0046    06F0            btfsc   txbuf,7         ; is bit out really low?
0182    0047    05CA            bsf     eeprom,do       ; no, set it high
0183    0048    0928            call    BITOUT          ; send the bit to serial EE
0184    0049    0370            rlf     txbuf           ; rotate txbuf left
0185    004A    02F1            decfsz  count           ; 8 bits done?
0186    004B    0A45            goto    TXLP            ; no - go again
0187    004C    0936            call    BITIN           ; read ack bit
0188    004D    06EA            btfsc   eeprom,di       ; check ack bit
```


© 1994 Microchip Technology Inc.

```
0189   004E  0525           bsf     port_a,ackf     ; set acknowledge fail LED if the
0190                                                 ; device did not pull data low
0191                         ;
0192   004F  0800           retlw   0
0193                 ;
0194                 ;*****************************************************************
0195                 ;       Receive data Routine
0196                 ;       This routine reads one byte of data from the part
0197                 ;       into the 'datai' register.  It then sends a high
0198                 ;       ack bit to indicate that no more data is to be read
0199                 ;*****************************************************************
0200                 RX
0201   0050  0C08           movlw   .8              ; set # bits to 8
0202   0051  0031           movwf   count
0203   0052  006D           clrf    datai           ; clear input register
0204   0053  0403           bcf     status,0        ; make sure carry bit is low

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 5

Line   PC    Opcode

0205   0054  036D  RXLP     rlf     datai           ; rotate datai 1 bit left
0206   0055  0936           call    BITIN           ; read a bit
0207   0056  06EA           btfsc   eeprom,di
0208   0057  050D           bsf     datai,0         ; set bit 0 if necessary
0209   0058  02F1           decfsz  count           ; 8 bits done?
0210   0059  0A54           goto    RXLP            ; no, do another
0211   005A  0800           retlw   0
0212                 ;
0213                 ;*****************************************************************
0214                 ;       Power up routine
0215                 ;       This is the program entry point, which in this case simply
0216                 ;       sets the port_a I/O lines and directs control to the
0217                 ;       Read routine.
0218                 ;*****************************************************************
0219                 PWRUP
0220   005B  0C00            movlw  b'00000000'
0221   005C  0005            tris   port_a           ; set port A as all output
0222   005D  0065            clrf   port_a           ; all output lines low
0223   005E  0A5F            goto   READ
0224
0225                 ;
0226                 ;*****************************************************************
0227                 ;       READ (sequential read routine)
0228                 ;
0229                 ;       This routine reads 8 consecutive addresses of the
0230                 ;       serial EE device starting at address 00 in the
0231                 ;       sequential read mode. Reading in this mode is more
0232                 ;       efficient than the random read mode as the control byte
0233                 ;       and address have to be sent only once at the beginning
0234                 ;       of the sequence.  As many consecutive addresses as
0235                 ;       needed can then be read from the part until a stop bit is
0236                 ;       sent.  In the read mode, the PIC16C54 must send the acknowledge
0237                 ;       bit after every 8 data bits from the device.  When the
0238                 ;       last byte needed has been read, then the controller will
0239                 ;       send a high acknowledge bit and then a stop bit to halt
0240                 ;       transmission from the device.
0241                 ;*****************************************************************
0242                 READ
0243                                                 ;
0244   005F  0425           bcf     port_a,ackf ; clear the ack fail LED if on
0245   0060  0C08           movlw   .8
0246   0061  0032           movwf   bcount      ; set number of bytes to read as 8
0247   0062  0CA0           movlw   b'10100000' ; set slave address and write mode
0248   0063  002F           movwf   slave
0249   0064  006C           clrf    addr        ; set starting address to 00
0250                                             ;
0251   0065  0901           call    BSTART      ; generate start bit
0252   0066  020F           movf    slave,w     ; get slave address
```


© 1994 Microchip Technology Inc.

```
0253   0067  0030          movwf   txbuf          ; into transmit buffer
0254   0068  0943          call    TX             ; and send it
0255   0069  020C          movf    addr,w         ; get word address

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:48:46 1994  Page 6

Line   PC    Opcode

0256   006A  0030          movwf   txbuf          ; into transmit buffer
0257   006B  0943          call    TX             ; and send it
0258   006C  0901          call    BSTART         ; generate start bit
0259   006D  0CA1          movlw   b'10100001'    ; get slave address and read mode
0260   006E  0030          movwf   txbuf          ; into transmit buffer
0261   006F  0943          call    TX             ; and transmit it
0262                       ;
0263   0070  0950  rbyte    call    RX             ; read 1 byte from device
0264   0071  02F2          decfsz  bcount         ; are all 8 bytes read?
0265   0072  0A77          goto    lowack         ; no, send low ack and do another
0266   0073  05CA          bsf     eeprom,do      ; yes, send high ack bit
0267   0074  0928          call    BITOUT         ; to stop transmission
0268   0075  0916          call    BSTOP          ; and send a stop bit
0269   0076  0A5F          goto    READ           ; start all over
0270
0271   0077  04CA  lowack   bcf     eeprom,do      ; send low ack bit
0272   0078  0928          call    BITOUT         ; to continue transmission
0273   0079  0A70          goto    rbyte          ; and read another byte
0274               ;
0275         0000  END
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN558

Using the 24XX65 and 24XX32 with Stand Alone PIC16C54 Code

USING THE SMART SERIAL™ SERIES

With the advent of CMOS silicon devices came the battery powered application. The battery powered application required more functionality and thus more power from the microcontroller which required more power from the batteries which cost more. Just as nature has a definite cycle, so it seems, does the portable application. Nowhere has this been more evident than in the areas of personal communication and data acquisition. Never before in the history of the industrial revolution have the type of applications seen emerging today been possible. Every man-made object that requires human or machine interface, has the potential to be controlled by embedded circuits and to be powered by the advanced technology batteries available today. Along with this increased functionality comes the requirement for MORE MEMORY. More in a smaller package, at a better price requiring less power. The majority of the hand-held embedded applications either use what little non-volatile memory that is available on the controllers or they use external devices. Parallel EEPROMs require too many I/O connects, which is the major source of device active current. Serial EEPROMs are typically the answer for these applications and the industry standard I²C™ is the industry leader (>70 % worldwide). The I²C standard specifies a maximum 16K bit address space. How does the personal communications system designer, who requires more memory, solve this problem?

THE I²C DILEMMA

The I²C serial bus has many advantages over other common serial interfaces for serial embedded devices. The I²C bus with level-triggered inputs offers better noise immunity over edge triggered technology. Opcodes are not needed to communicate with storage devices because all interfaces are intuitive and comparable to parallel devices. The I²C protocol assigns a slave address for each unique device or device family. Microchip Technology, and most non-volatile suppliers, use the slave address of 1010 for serial electrical erasable programmable read only memory (Serial EEPROM). The protocol also facilitates up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2. Herein lies the dilemma: with the advent of the more sophisticated personal communication devices such as cellular and full featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

Smart Serial Products

The Smart Serial concept grew from the industry need for increased memory requirements in I²C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands. Currently, Microchip Technology Inc. has the 24XX65 and 24XX32 devices available. All comments in this application note pertain primarily to the 24XX65 but the routines and general architectural comments apply to the 24XX32 as well.

The user should reference the individual data sheets for specific differences.

The Microchip 24XX65 is a serial memory device with 64K bits (8Kx8) capacity and additional patent pending unique features not found any where else. First let's look at the current I²C addressing scheme, the cascadable solution, and finally the Microchip total embedded systems solution.

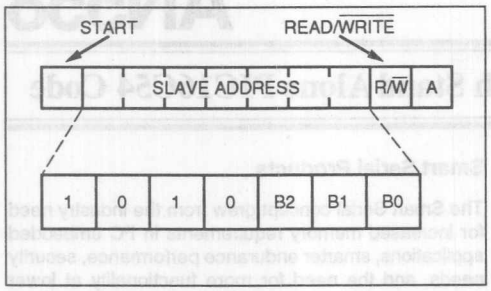
I²C ADDRESSING

The I²C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller, which controls the bus, generates the serial clock (SCL) and originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = 1010) is shown in Figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the 1010 identifying address plus the three block or chip select bits.

8

© 1994 Microchip Technology Inc.

FIGURE 1 - CONTROL BYTE ALLOCATION



All memory and peripheral devices on the I²C bus conform to this sequence for identification and selection. There are 128 assigned slave addresses in the standard protocol. There is a 10-bit extension to the protocol for 1024 additional slave addresses.

THE CASCADABLE SOLUTION

Cascading is an addressing scheme used in the 24164 (2Kx8 SEEPROM) to enable using more than the 16K limit set forth by the standard. In this method the A0, A1, and A2 pins are mapped into bits 2, 3, and 4 of the 8-bit slave address. This approach allows the system designer to use the non-standard 24164 to increase the total memory of a given memory subsystem. This solution is definitely workable, but does not offer the user the system design flexibility needed for the newer and more complex systems. Most system architects would prefer to have a linear address space for program and data memory. Programmers also find the linear solution more attractive. The overhead required for bank switching and chip selection usually requires additional overhead and hardware.

THE ULTIMATE SOLUTION BY MICROCHIP

Microchip Technology has designed an addressing scheme based on the standard I²C protocol and device addresses but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective, design. This enhanced addressing combined with the many advanced and patent pending features of the 24XX65 make the 24XX65 an exciting and innovative device. It is the first in a family of sophisticated **Smart Serial** EEPROMs from Microchip Technology Inc.

24XX65 Features

The 24XX65 has an advanced architecture with the following features:

- 15-bit, 2-byte address field (14 bit for the 24XX32)
- 4K-bit High Endurance Block - 1 Million E/W cycles typical (Fixed at the last 4k bit block in the array)
- Programmable write protect security features with up to a 15 blocks of 4K bits
- 8 byte by 8 line input write cache for I²C Fast Mode, burst mode capability, and use as a capture buffer

24XX65 Addressing

For the first byte or control byte, the 24XX65 adheres to the I²C protocol (reference Figure 2). This is the first byte received following the start condition from the master device. The control byte consists of a four-bit control code for the 24XX65 (this is assigned as 1010 binary for read and write operations). The next three bits of the control byte are the device select bits (A2, A1, A0). They are used by the master device to select which of the eight devices are to be accessed. These bits are in effect the three most significant bits of the word address. The last bit of the control byte defines the operation to be performed. When set to a "1" a read operation is selected, when set to a "0" a write operation is selected. The least significant 13 bits of the next 2 bytes define the address of the first byte within the 8K block. The most significant byte is transferred first. Following the start condition, the 24XX65 monitors the SDA bus, checking the device type identifier being transmitted, upon receiving a 1010 code and appropriate device select bits, the slave device outputs an acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24XX65 will select a read or write operation.

The addressing scheme uses the standard first byte slave address format of the I²C standard with the Microchip Technology-assigned 1010 slave address. The internal bus controller scans the next byte for bit 7 to be asserted indicating that a security operation is to take place. This will be further explained later. The remainder of the byte is composed of the most significant address bits. The next byte is the least significant address byte. See Figure 2 for graphical representation of this sequence. After receiving a valid 2 byte address and a stop bit, the 24XX65 will process this address according to bit "0" of the control byte and either wait for data to be written, if in a write sequence, or present the requested data if in a read sequence.

ADVANCED FEATURES

Programmable security

The 24XX65 has a sophisticated security mechanism by which selected blocks of memory may be write protected by the user. The write sequence includes a bit for enabling the security protection scheme, bit 7 of the first address byte. When this bit is set to a one, the first byte

© 1994 Microchip Technology Inc.

following the address during a write sequence defines the security block. This includes a pointer to the starting 4K block to be protected (the write address), a write/erase flag, and a non-zero four bit code for determining the number of 4K blocks to protect up to the maximum of 15 (60K). The 4K blocks must be contiguous. The high endurance block cannot be protected. In a normal application the security block or blocks would be set after all code or look-up table data has been finalized. THIS OPERATION CAN ONLY BE PERFORMED ONCE. (See Figures 2 and 3.)

Total Endurance™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate, for each byte in the application, not on the number of opcodes or control byte commands, and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. Applying the predictive data from the 24LC04B, which has similar characteristics, to the 4K high endurance block and assuming the following:

- a five-year life for a personal communication device
- an expected E/W cycles of 10 times per day
- a last number redial function of 11 bytes

Operational specifications:

Device	24XX65
Voltage	5
Temperature	25 C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4K HE block with 1 M E/W cycles typical, in this application, should yield the following results:

FIT	1.0
PPM	6
Time	5.0
Write cycles	18,250

FIGURE 2 - CONTROL BYTE ALLOCATION

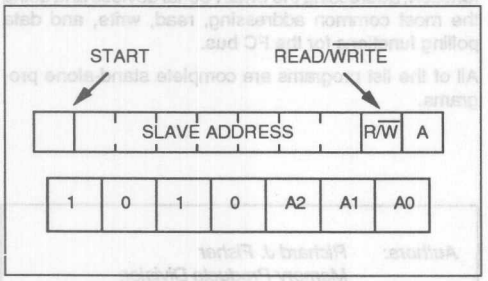
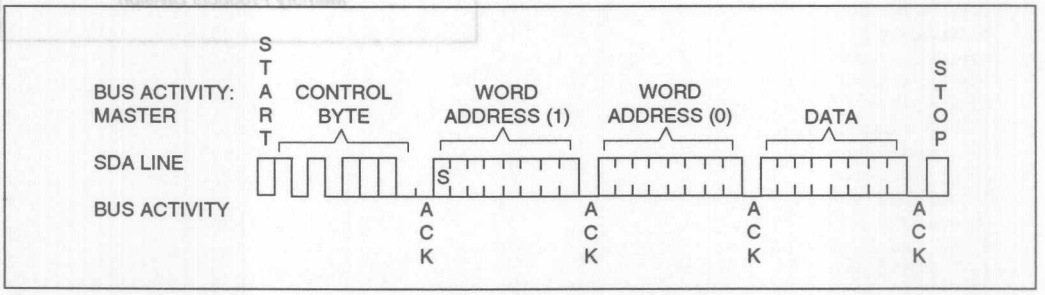


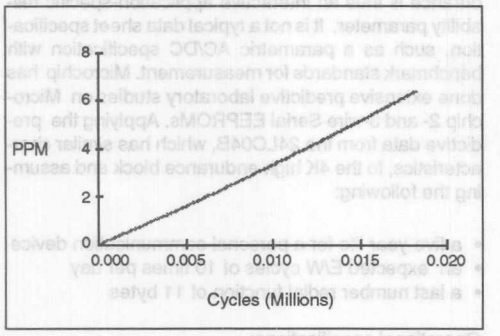
FIGURE 3 - BYTE WRITE



8

© 1994 Microchip Technology Inc.

FIGURE 4 - PPM RATE IN MILLIONS OF CYCLES



The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance see the related application notes AN537 and AN562 contained in this volume. (See Figure 4.)

64 Byte Write Cache

The cache is arranged in 8-byte pages by 8 lines each. This yields a total of 64 bytes. The interface to the 24XX65 supports both the standard 100KHz mode and FAST mode at 400KHz. The input cache can therefore support a burst write option of up to 64 bytes. When using the I²C protocol, an end of a page is defined by the transmission of a stop bit by the master. This sequence in the 24XX65 could be used to define pages from 8 to 64 bytes in length. The 8 byte by 8 line cache will roll over if a write is attempted past byte 8 of line 8, thus it can be used as a 64 byte capture or snapshot buffer. Each line is over written during a subsequent write to the same line. Faster memory and controller interfaces will become increasingly important in applications incorporating the ACCESS.bus™ interface standard.

Power Management

Increasingly, power management is becoming a predominant requirement for hand-held devices where a limited amount of power is available from the total power budget for a particular function. The 24XX65 has built-in power saving features such that the entire device is put into standby mode upon receiving a stop bit or an abort when in a read sequence, and after the completion of writing the data in the cache lines to the array when in an erase/write sequence. When the device is in standby mode, the only active circuit is the input circuit for the I²C clock. This yields a standby current that is the current consumption of this lone input and the normal leakage current of the silicon and typically will be less than 2 µA.

The 24XX65 is the first device available in the Smart Serial product line. The features and capabilities initiated with this device will become standard on all future Microchip Technology Inc. serials and some features will be enhanced, such as the security options.

Appendix A of this application note contains the required PIC16C54 assembly routines for setting the security function, addressing the Smart Serial devices and using the most common addressing, read, write, and data polling functions for the I²C bus.

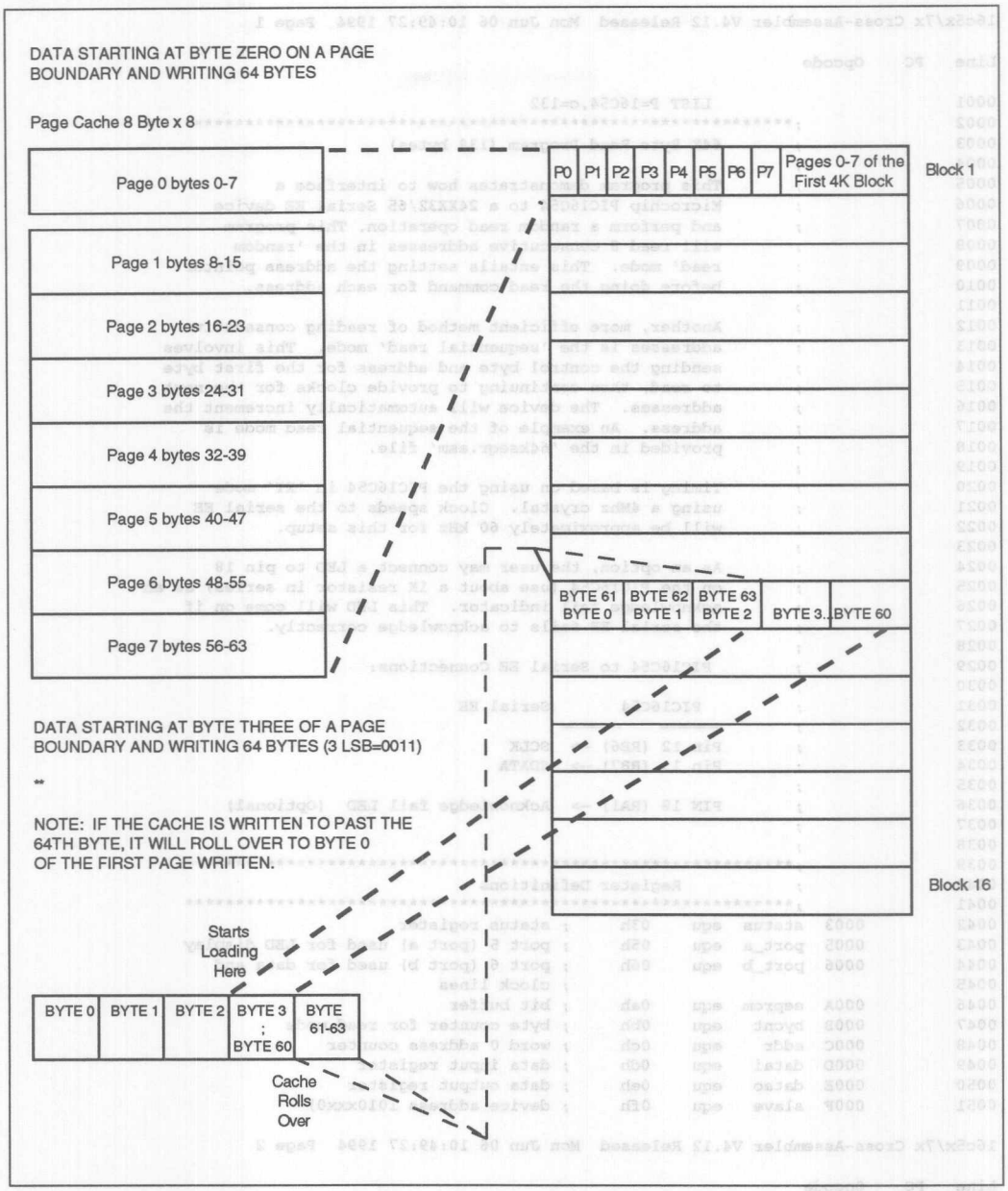
All of the list programs are complete stand-alone programs.

Authors: *Richard J. Fisher*
Memory Products Division

Bruce Negley
Memory Products Division

© 1994 Microchip Technology Inc.

FIGURE 5 - CACHE WRITE



8

APPENDIX A

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 1

Line   PC    Opcode

0001                      LIST P=16C54,c=132
0002                ;*****************************************************************
0003                ;       64K Byte Read Program (138 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 24XX32/65 Serial EE device
0007                ;       and perform a random read operation. This program
0008                ;       will read 8 consecutive addresses in the 'random
0009                ;       read' mode.  This entails setting the address pointer
0010                ;       before doing the read command for each address.
0011                ;
0012                ;       Another, more efficient method of reading consecutive
0013                ;       addresses is the 'sequential read' mode.  This involves
0014                ;       sending the control byte and address for the first byte
0015                ;       to read, then continuing to provide clocks for the next
0016                ;       addresses.  The device will automatically increment the
0017                ;       address.  An example of the sequential read mode is
0018                ;       provided in the '64kseqr.asm' file.
0019                ;
0020                ;       Timing is based on using the PIC16C54 in 'XT' mode
0021                ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0022                ;       will be approximately 60 kHz for this setup.
0023                ;
0024                ;       As an option, the user may connect a LED to pin 18
0025                ;       on the PIC16C54 (use about a 1K resistor in series) as an
0026                ;       acknowledge fail indicator.  This LED will come on if
0027                ;       the serial EE fails to acknowledge correctly.
0028                ;
0029                ;        PIC16C54 to Serial EE Connections:
0030                ;
0031                ;          PIC16C54        Serial EE
0032                ;          --------        ---------
0033                ;       Pin 12 (RB6) ->  SCLK
0034                ;       Pin 13 (RB7) ->  SDATA
0035                ;
0036                ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional)
0037                ;
0038                ;
0039                ;*****************************************************************
0040                ;             Register Definitions
0041                ;*****************************************************************
0042          0003  status  equ     03h     ; status register
0043          0005  port_a  equ     05h     ; port 5 (port a) used for LED display
0044          0006  port_b  equ     06h     ; port 6 (port b) used for data and
0045                                        ; clock lines
0046          000A  eeprom  equ     0ah     ; bit buffer
0047          000B  bycnt   equ     0bh     ; byte counter for read mode
0048          000C  addr    equ     0ch     ; word 0 address counter
0049          000D  datai   equ     0dh     ; data input register
0050          000E  datao   equ     0eh     ; data output register
0051          000F  slave   equ     0fh     ; device address 1010xxx0)

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 2

Line   PC    Opcode

0052          0010  txbuf   equ     10h     ; transmit buffer
0053          0011  count   equ     11h     ; bit counter
0054          0012  bcount  equ     12h     ; byte counter
0055          0015  loops   equ     15h     ; delay loop counter
0056          0016  loops2  equ     16h     ; delay loop counter
0057          0017  addr1   equ     17h     ; word 1 address counter
0058                ;*****************************************************************
```


© 1994 Microchip Technology Inc.

```
0059                    ;           Bit Definitions
0060                    ;*************************************************************
0061            0007    di      equ     7       ; eeprom input bit
0062            0006    do      equ     6       ; eeprom output bit
0063            0007    sdata   equ     7       ; serial EE data line (port_b,pin 13)
0064            0006    sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0065            0001    ackf    equ     1       ; acknowledge fail LED (port_a)
0066                    ;*************************************************************
0067                    ;
0068            0000            org     01ffh   ; set reset vector
0069    01FF    0A6B            goto    PWRUP
0070            0000            org     000h    ;
0071    0000    0A6B            goto    PWRUP
0072                    ;
0073                    ;*************************************************************
0074                    ;       DELAY ROUTINE
0075                    ;       This routine takes the value in 'loops'
0076                    ;       and multiplies it times 1 millisecond to
0077                    ;       determine delay time.
0078                    ;*************************************************************
0079                    WAIT
0080                    ;
0081    0001    0C6E    top     movlw   .110            ; timing adjustment variable
0082    0002    0036            movwf   loops2
0083    0003    0000    top2    nop                     ; sit and wait
0084    0004    0000            nop
0085    0005    0000            nop
0086    0006    0000            nop
0087    0007    0000            nop
0088    0008    0000            nop
0089    0009    02F6            decfsz  loops2          ; inner loops complete?
0090    000A    0A03            goto    top2            ; no, go again
0091                                                    ;
0092    000B    02F5            decfsz  loops           ; outer loops complete?
0093    000C    0A01            goto    top             ; no, go again
0094    000D    0800            retlw   0               ; yes, return from sub
0095                    ;
0096                    ;*************************************************************
0097                    ;       Start Bit Subroutine
0098                    ;       this routine generates a start bit
0099                    ;       (Low going data line while clock is high)
0100                    ;*************************************************************
0101                    ;
0102                    BSTART

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 3

Line    PC      Opcode

0103    000E    05E6            bsf     port_b,sdata    ; make sure data is high
0104    000F    0C3F            movlw   b'00111111'
0105    0010    0006            tris    port_b          ; set data and clock lines for output
0106    0011    04C6            bcf     port_b,sclk     ; make sure clock is low
0107    0012    0000            nop
0108    0013    05C6            bsf     port_b,sclk     ; set clock high
0109    0014    0000            nop
0110    0015    0000            nop
0111    0016    0000            nop
0112    0017    0000            nop
0113    0018    0000            nop
0114    0019    04E6            bcf     port_b,sdata    ; data line goes low during
0115                                                    ; high clock for start bit
0116    001A    0000            nop
0117    001B    0000            nop
0118    001C    0000            nop
0119    001D    0000            nop
0120    001E    0000            nop                     ; timing adjustment
0121    001F    04C6            bcf     port_b,sclk     ; start clock train
0122    0020    0000            nop
```

8

© 1994 Microchip Technology Inc.

```
0123   0021  0000          nop
0124   0022  0800          retlw   0
0125                       ;
0126                ;*************************************************************
0127                ;       Stop Bit Subroutine
0128                ;       This routine generates a stop bit
0129                ;         (High going data line while clock is high)
0130                ;*************************************************************
0131                BSTOP
0132   0023  04E6          bcf     port_b,sdata    ; make sure data line is low
0133   0024  0C3F          movlw   b'00111111'     ;
0134   0025  0006          tris    port_b          ; set data/clock lines as outputs
0135   0026  04E6          bcf     port_b,sdata    ; make sure data line is low
0136   0027  0000          nop
0137   0028  0000          nop
0138   0029  0000          nop
0139   002A  05C6          bsf     port_b,sclk     ; set clock high
0140   002B  0000          nop
0141   002C  0000          nop
0142   002D  0000          nop
0143   002E  05E6          bsf     port_b,sdata    ; data goes high while clock high
0144                                                ; for stop bit
0145   002F  0000          nop
0146   0030  0000          nop
0147   0031  04C6          bcf     port_b,sclk     ; set clock low again
0148   0032  0000          nop
0149   0033  0000          nop
0150   0034  0000          nop
0151   0035  0800          retlw   0
0152                ;
0153                ;*************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 4

Line   PC    Opcode

0154                ;       BITOUT routine takes the bit of data in 'do' and
0155                ;       transmits it to the serial EE device
0156                ;*************************************************************
0157                BITOUT
0158   0036  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0159   0037  0006          tris    port_b
0160   0038  07CA          btfss   eeprom,do       ; check for state of data bit to xmit
0161   0039  0A3C          goto    bitlow          ; low? go set data line low
0162   003A  05E6          bsf     port_b,sdata    ; high? set data line high
0163   003B  0A3D          goto    clkout          ; go toggle the clock
0164
0165   003C  04E6  bitlow   bcf     port_b,sdata    ; output a low bit
0166   003D  05C6  clkout   bsf     port_b,sclk     ; set clock line high
0167   003E  0000          nop
0168   003F  0000          nop
0169   0040  0000          nop
0170   0041  0000          nop
0171   0042  04C6          bcf     port_b,sclk     ; return clock line low
0172   0043  0800          retlw   0
0173                ;
0174                ;*************************************************************
0175                ;       BITIN routine reads one bit of data from the
0176                ;       serial EE device and stores it in the bit 'di'
0177                ;*************************************************************
0178                BITIN
0179   0044  05EA          bsf     eeprom,di       ; assume input bit is high
0180   0045  0CBF          movlw   b'10111111'     ; make sdata an input line
0181   0046  0006          tris    port_b
0182   0047  05C6          bsf     port_b,sclk     ; set clock line high
0183   0048  0000          nop                     ; just sit here a sec
0184   0049  0000          nop
0185   004A  0000          nop
0186   004B  0000          nop
```


© 1994 Microchip Technology Inc.

```
0187   004C  0000          nop                     ;
0188   004D  07E6          btfss   port_b,sdata    ; read the data bit
0189   004E  04EA          bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0190   004F  04C6          bcf     port_b,sclk     ; set clock line low
0191                                               ;
0192   0050  0800          retlw   0               ;
0193                ;
0194                ;*****************************************************************
0195                ;       Transmit Data Subroutine
0196                ;       This routine takes the byte of data stored in the
0197                ;       'datao' register and transmits it to the serial EE device.
0198                ;       It will then send 1 more clock to the serial EE for the
0199                ;       acknowledge bit.  If the ack bit from the part was low
0200                ;       then the transmission was successful.  If it is high, then
0201                ;       the device did not send a proper ack bit and the ack
0202                ;       fail LED will be turned on.
0203                ;*****************************************************************
0204                TX

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 5

Line   PC    Opcode

0205   0051  0C08          movlw   .8
0206   0052  0031          movwf   count           ; set the #bits to 8
0207                       ;
0208                TXLP
0209   0053  04CA          bcf     eeprom,do       ; assume bit out is low
0210   0054  06F0          btfsc   txbuf,7         ; is bit out really low?
0211   0055  05CA          bsf     eeprom,do       ; no, set it high
0212   0056  0936          call    BITOUT          ; send the bit to serial EE
0213   0057  0370          rlf     txbuf           ; rotate txbuf left
0214   0058  02F1          decfsz  count           ; 8 bits done?
0215   0059  0A53          goto    TXLP            ; no - go again
0216   005A  0944          call    BITIN           ; read ack bit
0217   005B  06EA          btfsc   eeprom,di       ; check ack bit
0218   005C  0525          bsf     port_a,ackf     ; set acknowledge fail LED if the
0219                                               ; device did not pull data low
0220                       ;
0221   005D  0800          retlw   0
0222                ;
0223                ;*****************************************************************
0224                ;       Receive data routine
0225                ;       This routine reads one byte of data from the part
0226                ;       into the 'datai' register.  It then sends a high
0227                ;       ack bit to indicate that no more data is to be read
0228                ;*****************************************************************
0229                RX
0230   005E  006D          clrf    datai           ; clear input buffer
0231   005F  0C08          movlw   .8              ; set # bits to 8
0232   0060  0031          movwf   count
0233   0061  0403          bcf     status,0        ; make sure carry bit is low
0234   0062  036D  RXLP    rlf     datai           ; rotate datai 1 bit left
0235   0063  0944          call    BITIN           ; read a bit
0236   0064  06EA          btfsc   eeprom,di
0237   0065  050D          bsf     datai,0         ; set bit 0 if necessary
0238   0066  02F1          decfsz  count           ; 8 bits done?
0239   0067  0A62          goto    RXLP            ; no, do another
0240   0068  05CA          bsf     eeprom,do       ; set ack bit = 1
0241   0069  0936          call    BITOUT          ; to finish transmission
0242   006A  0800          retlw   0
0243                ;
0244                ;*****************************************************************
0245                ;       Power up routine
0246                ;       This is the program entry point.  I/O line status is
0247                ;       set for port A here.
0248                ;*****************************************************************
0249                PWRUP
0250   006B  0C00           movlw  b'00000000'
```

8

© 1994 Microchip Technology Inc.

```
0251   006C  0005            tris   port_a          ; set port A as all output
0252   006D  0065            clrf   port_a          ; all output lines low
0253   006E  0A6F            goto   READ
0254                 ;
0255                 ;*****************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:27 1994  Page 6

Line   PC    Opcode

0256                 ;       READ (read routine)
0257                 ;       This routine reads 8 consecutive addresses of the
0258                 ;       serial EE device starting at address 00 in the
0259                 ;       random access mode (non-sequential read). Reading
0260                 ;       the device using the random access mode
0261                 ;       requires that the address pointer be set for every
0262                 ;       byte before the read takes place. The address pointer
0263                 ;       is set by sending a 'write mode' control byte with the
0264                 ;       address of the location to read.
0265                 ;*****************************************************************
0266                 READ
0267                 ;
0268   006F  0425            bcf    port_a,ackf ; clear the ack fail LED if on
0269   0070  0C08            movlw  .8          ; set number of bytes to read as 8
0270   0071  0032            movwf  bcount      ;
0271   0072  0077            clrf   addr1       ; set starting high address byte to 00
0272   0073  006C            clrf   addr        ; set starting low address byte to 00
0273                                             ;
0274   0074  090E  rbyte     call   BSTART      ; generate start bit
0275                                             ;
0276                                             ; now send the write control byte and
0277                                             ; address to set the pointer
0278                                             ;
0279   0075  0CA0            movlw  b'10100000' ; get slave address (write mode)
0280   0076  0030            movwf  txbuf       ; into transmit buffer
0281   0077  0951            call   TX          ; and send it
0282   0078  0217            movf   addr1,w     ; get word 1 address
0283   0079  0030            movwf  txbuf       ; into transmit buffer
0284   007A  0951            call   TX          ; and send it
0285   007B  020C            movf   addr,w      ; get word 0 address
0286   007C  0030            movwf  txbuf       ; into transmit buffer
0287   007D  0951            call   TX          ; and send it
0288                                             ;
0289                                             ; now read one byte from the part
0290                                             ;
0291   007E  090E            call   BSTART      ; generate start bit
0292   007F  0CA1            movlw  b'10100001' ; get slave address and read mode
0293   0080  0030            movwf  txbuf       ; into transmit buffer
0294   0081  0951            call   TX          ; and transmit it
0295   0082  095E            call   RX          ; read 1 byte from serial EE
0296   0083  0923            call   BSTOP       ; send stop bit to end transmission
0297   0084  02AC            incf   addr        ; add 1 to address counter
0298   0085  02F2            decfsz bcount      ; yes, are all 8 bytes read?
0299   0086  0A74            goto   rbyte       ; no, do another byte
0300
0301   0087  0CFF            movlw  .255        ; long delay for scope
0302   0088  0035            movwf  loops       ; trigger purposes only
0303   0089  0901            call   wait
0304   008A  0A6F            goto   READ        ; yes, start all over
0305                 ;
0306         0000  END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 1

Line   PC    Opcode

0001                LIST P=16C54,c=132
0002               ;*************************************************************
0003               ;       64K Byte Write Program (127 bytes)
0004               ;
0005               ;       This program demonstrates how to interface a
0006               ;       Microchip PIC16C54 to the 24XX32/65 Serial EE device
0007               ;       and perform a byte write operation on 8 consecutive
0008               ;       addresses.
0009               ;
0010               ;       After each byte is written, time must be given to the
0011               ;       device for it to complete the write cycle before
0012               ;       the next command can be sent.  The easiest solution
0013               ;       is to consult the data book for the maximum write
0014               ;       cycle time and just wait that long before the next
0015               ;       command is sent(10ms for this device).  This program
0016               ;       demonstrates that solution.
0017               ;
0018               ;       Another, more efficient method of determining when the
0019               ;       write cycle is complete is called 'data polling.'  This
0020               ;       method is demonstrated in the program "64kdpoll."
0021               ;
0022               ;       As an option, the user may connect a LED to pin 18
0023               ;       on the PIC16C54 (use about a 1K resistor in series) as an
0024               ;       acknowledge fail indicator.  This LED will come on if
0025               ;       the serial EE fails to acknowledge correctly.
0026               ;
0027               ;       Timing is based on using the PIC16C54 in 'XT' mode
0028               ;       using a 4MHz crystal.  Clock speeds to the serial EE
0029               ;       will be approximately 60 kHz for this setup.
0030               ;
0031               ;        PIC16C54 to Serial EE Connections:
0032               ;
0033               ;        PIC16C54          Serial EE
0034               ;        --------          ---------
0035               ;       Pin 12 (RB6) -->  SCLK
0036               ;       Pin 13 (RB7) -->  SDATA
0037               ;
0038               ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
0039               ;
0040               ;*************************************************************
0041               ;             Register Definitions
0042               ;*************************************************************
0043        0005   port_a  equ     5h      ; port 5 (port_a) used for LEDs
0044        0006   port_b  equ     6h      ; port 6 (port b) used for data and
0045                                       ; clock lines
0046        000A   eeprom  equ     0ah     ; bit buffer
0047        000B   bycnt   equ     0bh     ; byte counter for read mode
0048        000C   addr    equ     0ch     ; word 0 address counter
0049        000D   datai   equ     0dh     ; data input register
0050        000E   datao   equ     0eh     ; data output register
0051        000F   slave   equ     0fh     ; device address (1010xxx0)

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 2

Line   PC    Opcode

0052        0010   txbuf   equ     10h     ; transmit buffer
0053        0011   count   equ     11h     ; bit counter
0054        0012   bcount  equ     12h     ; byte counter
0055        0015   loops   equ     15h     ; delay loop counter
0056        0016   loops2  equ     16h     ; delay loop counter
0057        0017   addr1   equ     17h     ; word 1 address counter
0058               ;*************************************************************
0059               ;             Bit Definitions
0060               ;*************************************************************
```


© 1994 Microchip Technology Inc.

```
0061          0007  di      equ     7       ; eeprom input bit
0062          0006  do      equ     6       ; eeprom output bit
0063          0007  sdata   equ     7       ; serial EE data line (port_b,pin 13)
0064          0006  sclk    equ     6       ; serial EE clock line (port_b,pin 12)
0065          0001  ackf    equ     1       ; acknowledge fail LED (port_a,pin 18)
0066                ;*************************************************************
0067                ;
0068                ;
0069          0000          org     01ffh   ; set reset vector
0070    01FF  0A5E          goto    PWRUP
0071          0000          org     000h    ;
0072    0000  0A5E          goto    PWRUP
0073                ;
0074                ;*************************************************************
0075                ;       DELAY ROUTINE
0076                ;       This routine takes the value in 'loops'
0077                ;       and multiplies it times 1 millisecond to
0078                ;       determine delay time.
0079                ;*************************************************************
0080                WAIT
0081                ;
0082    0001  0C6E  top       movlw   .110            ; timing adjustment variable
0083    0002  0036            movwf   loops2
0084    0003  0000  top2      nop                     ; sit and wait
0085    0004  0000            nop
0086    0005  0000            nop
0087    0006  0000            nop
0088    0007  0000            nop
0089    0008  0000            nop
0090    0009  02F6            decfsz  loops2          ; inner loops complete?
0091    000A  0A03            goto    top2            ; no, go again
0092                                                  ;
0093    000B  02F5            decfsz  loops           ; outer loops complete?
0094    000C  0A01            goto    top             ; no, go again
0095    000D  0800            retlw   0               ; yes, return from sub
0096                ;
0097                ;*************************************************************
0098                ;       Start Bit Subroutine
0099                ;       this routine generates a start bit
0100               ;       (Low going data line while clock is high)
0101               ;*************************************************************
0102               ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 3

Line   PC    Opcode

0103               BSTART
0104    000E  05E6            bsf     port_b,sdata    ; make sure data is high
0105    000F  0C3F            movlw   b'00111111'
0106    0010  0006            tris    port_b          ; set data and clock lines for output
0107    0011  04C6            bcf     port_b,sclk     ; make sure clock is low
0108    0012  0000            nop
0109    0013  05C6            bsf     port_b,sclk     ; set clock high
0110    0014  0000            nop
0111    0015  0000            nop
0112    0016  0000            nop
0113    0017  0000            nop
0114    0018  0000            nop
0115    0019  04E6            bcf     port_b,sdata    ; data line goes low during
0116                                                  ; high clock for start bit
0117    001A  0000            nop
0118    001B  0000            nop
0119    001C  0000            nop
0120    001D  0000            nop
0121    001E  0000            nop                     ; timing adjustment
0122    001F  04C6            bcf     port_b,sclk     ; start clock train
0123    0020  0000            nop
0124    0021  0000            nop
```


© 1994 Microchip Technology Inc.

```
0125   0022  0800          retlw   0
0126                 ;
0127                 ;*************************************************************
0128                 ;       Stop Bit Subroutine
0129                 ;       This routine generates a stop bit
0130                 ;       (High going data line while clock is high)
0131                 ;*************************************************************
0132                 BSTOP
0133   0023  0C3F          movlw   b'00111111'     ;
0134   0024  0006          tris    port_b          ; set data/clock lines as outputs
0135   0025  04E6          bcf     port_b,sdata    ; make sure data line is low
0136   0026  0000          nop
0137   0027  0000          nop
0138   0028  0000          nop
0139   0029  05C6          bsf     port_b,sclk     ; set clock high
0140   002A  0000          nop
0141   002B  0000          nop
0142   002C  0000          nop
0143   002D  05E6          bsf     port_b,sdata    ; data goes high while clock high
0144                                                ; for stop bit
0145   002E  0000          nop
0146   002F  0000          nop
0147   0030  04C6          bcf     port_b,sclk     ; set clock low again
0148   0031  0000          nop
0149   0032  0000          nop
0150   0033  0000          nop
0151   0034  0800          retlw   0
0152                 ;
0153                 ;*************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 4

Line   PC    Opcode

0154                 ;       BITOUT routine takes one bit of data in 'do' and
0155                 ;       transmits it to the serial EE device
0156                 ;*************************************************************
0157                 BITOUT
0158   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0159   0036  0006          tris    port_b
0160   0037  07CA          btfss   eeprom,do       ; check for state of data bit to xmit
0161   0038  0A3B          goto    bitlow          ;
0162   0039  05E6          bsf     port_b,sdata    ; set data line high
0163   003A  0A3C          goto    clkout          ; go toggle the clock
0164
0165   003B  04E6  bitlow  bcf     port_b,sdata    ; output a low bit
0166   003C  05C6  clkout  bsf     port_b,sclk     ; set clock line high
0167   003D  0000          nop
0168   003E  0000          nop
0169   003F  0000          nop
0170   0040  0000          nop
0171   0041  04C6          bcf     port_b,sclk     ; return clock line low
0172   0042  0800          retlw   0
0173                 ;
0174                 ;*************************************************************
0175                 ;       BITIN routine reads one bit of data from the
0176                 ;       serial EE device and stores it in 'di'
0177                 ;*************************************************************
0178                 BITIN
0179   0043  05EA          bsf     eeprom,di       ; assume input bit is high
0180   0044  0CBF          movlw   b'10111111'     ; make sdata an input line
0181   0045  0006          tris    port_b
0182   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
0183   0047  05C6          bsf     port_b,sclk     ; set clock line high
0184   0048  0000          nop                     ; just sit here a sec
0185   0049  0000          nop
0186   004A  0000          nop
0187   004B  0000          nop
0188   004C  0000          nop                     ;
```

8

```
0189   004D  07E6          btfss   port_b,sdata    ; read the data bit
0190   004E  04EA          bcf     eeprom,di       ; input bit was low
0191   004F  04C6          bcf     port_b,sclk     ; set clock line low
0192                                               ;
0193   0050  0800          retlw   0               ;
0194                ;
0195                ;*****************************************************************
0196                ;       Transmit Data Subroutine
0197                ;       This routine takes the byte of data stored in the
0198                ;       'datao' register and transmits it to the serial EE device.
0199                ;       It will then send 1 more clock to the serial EE for the
0200                ;       acknowledge bit.  If the ack bit from the part was low
0201                ;       then the transmission was successful.  If it is high, then
0202                ;       the device did not send a proper ack bit and the ack
0203                ;       fail LED will be turned on.
0204                ;*****************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 5

Line   PC    Opcode

0205                TX
0206   0051  0C08          movlw   .8
0207   0052  0031          movwf   count           ; set the #bits to 8
0208                       ;
0209                TXLP
0210   0053  04CA          bcf     eeprom,do       ; assume bit out is low
0211   0054  06F0          btfsc   txbuf,7         ; is bit out really low?
0212   0055  05CA          bsf     eeprom,do       ; otherwise data bit =1
0213   0056  0935          call    BITOUT          ; serial data out
0214   0057  0370          rlf     txbuf           ; rotate txbuf left
0215   0058  02F1          decfsz  count           ; 8 bits done?
0216   0059  0A53          goto    TXLP            ; no - go again
0217   005A  0943          call    BITIN           ; read ack bit
0218   005B  06EA          btfsc   eeprom,di       ; check ack bit
0219   005C  0525          bsf     port_a,ackf     ; set acknowledge fail LED if the
0220                       ;
0221   005D  0800          retlw   0
0222                ;
0223                ;*****************************************************************
0224                ;       Power up routine
0225                ;       This is the program entry point, which in this case simply
0226                ;       sets the port_a I/O lines and directs control to the
0227                ;       byte write routine.
0228                ;*****************************************************************
0229                PWRUP
0230   005E  0C00           movlw  b'00000000'
0231   005F  0005           tris   port_a          ; set port A as all output
0232   0060  0065           clrf   port_a          ; all output lines low
0233   0061  0A62           goto   WRBYTE
0234
0235                ;*****************************************************************
0236                ;       Byte Write Routine
0237                ;       This routine writes the data in "datao" to
0238                ;       8 consecutive bytes in the serial EE device starting
0239                ;       at address 00.  This routine waits 10mS after every
0240                ;       byte to give the device time to do the write.  This
0241                ;       program repeats forever.
0242                ;*****************************************************************
0243                ;
0244                WRBYTE
0245                ;
0246   0062  0065           clrf   port_a      ; clear all LEDs
0247   0063  0CA0           movlw  b'10100000' ; set slave address and write mode
0248   0064  002F           movwf  slave
0249   0065  0C55           movlw  b'01010101' ; set data to write as 55h
0250   0066  002E           movwf  datao
0251                ;
0252   0067  0C08           movlw  .8          ; set number of bytes
```


© 1994 Microchip Technology Inc.

```
0253   0068  0032          movwf   bcount      ; to write as to 8
0254   0069  0077          clrf    addr1       ; set high address byte to 00
0255   006A  006C          clrf    addr        ; set low address byte to 00

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:40 1994  Page 6

Line   PC    Opcode

0256                              ;
0257   006B  090E  byte    call    BSTART      ; generate start bit
0258   006C  020F          movf    slave,w     ; move slave address
0259   006D  0030          movwf   txbuf       ; into transmit buffer
0260   006E  0951          call    TX          ; and send it
0261   006F  0217          movf    addr1,w     ; move word 1 address
0262   0070  0030          movwf   txbuf       ; into transmit buffer
0263   0071  0951          call    TX          ; and send it
0264   0072  020C          movf    addr,w      ; move word 0 address
0265   0073  0030          movwf   txbuf       ; into transmit buffer
0266   0074  0951          call    TX          ; and send it
0267   0075  020E          movf    datao,w     ; move data byte
0268   0076  0030          movwf   txbuf       ; to transmit buffer
0269   0077  0951          call    TX          ; and transmit it
0270   0078  0923          call    BSTOP       ; generate stop bit
0271                              ;
0272   0079  0C0A          movlw   .10
0273   007A  0035          movwf   loops       ; set delay time to give
0274   007B  0901          call    WAIT        ; 10 ms wait after every byte
0275   007C  02AC          incf    addr        ; add 1 to low address counter
0276   007D  02F2          decfsz  bcount      ; all 8 bytes written?
0277   007E  0A6B          goto    byte        ; no, do another byte
0278                              ;
0279   007F  0A62          goto    wrbyte      ; start over
0280                 ;
0281                 ;
0282         0000  END
```

8

© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       64K Byte Write With Data Polling Program (133 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 24XX32/65 Serial EE device.
0007                ;       This program performs a byte write operation on 8
0008                ;       consecutive addresses using the data polling method
0009                ;       to determine when the write cycle is complete.
0010                ;
0011                ;       When writing to a serial E≤ device, there are 2
0012                ;       ways to handle the internal timed write cycle
0013                ;       time of the device.  The simplest method is
0014                ;       simply to wait until the maximum cycle time
0015                ;       is exceeded before attempting another command.
0016                ;       The other, more efficient method is known as "data
0017                ;       polling."  Data polling is done by sending a start
0018                ;       bit and control byte to the part after the write
0019                ;       cycle has been initiated by a stop bit.  If the ack bit
0020                ;       is low, then the device is through writing, otherwise
0021                ;       the the sequence is repeated.  If no low acknowledge
0022                ;       is found within 40 attempts (about 10 milliseconds)
0023                ;       then the routine times out and sets the timeout
0024                ;       LED (pin 1) high.
0025                ;
0026                ;       As an option, the user can connect a LED to pin 1
0027                ;       on the PIC16C54 (use about a 1K resistor in series) as a
0028                ;       timeout indicator.  This LED will come on if the
0029                ;       data polling is unsuccessful and the device being
0030                ;       programmed does not respond.  This can be tested by
0031                ;       simply running the program with no part in the socket.
0032                ;
0033                ;       Timing is based on using the PIC16C54 in 'XT' mode
0034                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0035                ;       will be approximately 40 kHz for this setup.
0036                ;
0037                ;        PIC16C54 to Serial EE Connections:
0038                ;
0039                ;        PIC16C54       Serial EE
0040                ;        --------       ---------
0041                ;       Pin 12 (RB6) ->  SCLK
0042                ;       Pin 13 (RB7) ->  SDATA
0043                ;
0044                ;       Pin 1  (RA2) ->  Write cycle timeout fail LED (optional)
0045                ;
0046                ;*************************************************************
0047                ;           Register Definitions
0048                ;*************************************************************
0049         0005   port_a  equ     5h      ; port 5 (port_a) used LEDs
0050         0006   port_b  equ     6h      ; port 6 (port_b) used for data and
0051                                        ; clock lines

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 2

Line   PC    Opcode

0052         000A   eeprom  equ     0ah     ; bit buffer
0053         000B   bycnt   equ     0bh     ; byte counter for read mode
0054         000C   addr    equ     0ch     ; address counter
0055         000D   datai   equ     0dh     ; data input register
0056         000E   datao   equ     0eh     ; data output register
0057         000F   slave   equ     0fh     ; device address (1010xxx0)
0058         0010   txbuf   equ     10h     ; transmit buffer
0059         0011   count   equ     11h     ; bit counter
0060         0012   bcount  equ     12h     ; byte counter
```


© 1994 Microchip Technology Inc.

```
0061          0015  loops   equ    15h     ; delay loop counter
0062          0016  loops2  equ    16h     ; delay loop counter
0063          0017  pollcnt equ    17h     ; data polling counter
0064          0018  addr1   equ    18h     ; word 1 address counter
0065                ;*************************************************************
0066                ;           Bit Definitions
0067                ;*************************************************************
0068          0007  di      equ    7       ; eeprom input bit
0069          0006  do      equ    6       ; eeprom output bit
0070          0007  sdata   equ    7       ; serial EE data line (port_b,pin 13)
0071          0006  sclk    equ    6       ; serial EE clock line (port_b,pin 12)
0072          0002  timeout equ    2       ; write cycle timeout fail LED, port_a (pin 1)
0073          0001  ackf    equ    1       ; acknowledge fail LED, port_a (pin 18)
0074                ;*************************************************************
0075                ;
0076          0000          org    01ffh   ; set reset vector
0077    01FF  0A5C          goto   PWRUP
0078          0000          org    000h    ;
0079    0000  0A5C          goto   PWRUP
0080                ;
0081                ;*************************************************************
0082                ;       DELAY ROUTINE
0083                ;       This routine takes the value in 'loops'
0084                ;       and multiplies it times 1 millisecond to
0085                ;       determine delay time.
0086                ;*************************************************************
0087                WAIT
0088                ;
0089    0001  0C6E  top     movlw   .110            ; timing adjustment variable
0090    0002  0036          movwf   loops2
0091    0003  0000  top2    nop                     ; sit and wait
0092    0004  0000          nop
0093    0005  0000          nop
0094    0006  0000          nop
0095    0007  0000          nop
0096    0008  0000          nop
0097    0009  02F6          decfsz  loops2          ; inner loops complete?
0098    000A  0A03          goto    top2            ; no, go again
0099                                                ;
0100    000B  02F5          decfsz  loops           ; outer loops complete?
0101    000C  0A01          goto    top             ; no, go again
0102    000D  0800          retlw   0               ; yes, return from sub

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 3

Line    PC    Opcode

0103                ;
0104                ;*************************************************************
0105                ;       Start Bit Subroutine
0106                ;       this routine generates a start bit
0107                ;       (Low going data line while clock is high)
0108                ;*************************************************************
0109                ;
0110                BSTART
0111    000E  05E6          bsf     port_b,sdata    ; make sure data is high
0112    000F  0C3F          movlw   b'00111111'
0113    0010  0006          tris    port_b          ; set data and clock lines for output
0114    0011  04C6          bcf     port_b,sclk     ; make sure clock is low
0115    0012  0000          nop
0116    0013  05C6          bsf     port_b,sclk     ; set clock high
0117    0014  0000          nop
0118    0015  0000          nop
0119    0016  0000          nop
0120    0017  0000          nop
0121    0018  0000          nop
0122    0019  04E6          bcf     port_b,sdata    ; data line goes low during
0123                                                ; high clock for start bit
0124    001A  0000          nop
```

8

© 1994 Microchip Technology Inc.

```
0125   001B  0000           nop
0126   001C  0000           nop
0127   001D  0000           nop
0128   001E  0000           nop                ; timing adjustment
0129   001F  04C6           bcf     port_b,sclk    ; start clock train
0130   0020  0000           nop
0131   0021  0000           nop
0132   0022  0800           retlw   0
0133                        ;
0134                ;*****************************************************
0135                ;   Stop Bit Subroutine
0136                ;   This routine generates a stop bit
0137                ;   (High going data line while clock is high)
0138                ;*****************************************************
0139                BSTOP
0140   0023  0C3F           movlw   b'00111111'    ;
0141   0024  0006           tris    port_b         ; set data/clock lines as outputs
0142   0025  04E6           bcf     port_b,sdata   ; make sure data line is low
0143   0026  0000           nop
0144   0027  0000           nop
0145   0028  0000           nop
0146   0029  05C6           bsf     port_b,sclk    ; set clock high
0147   002A  0000           nop
0148   002B  0000           nop
0149   002C  0000           nop
0150   002D  05E6           bsf     port_b,sdata   ; data goes high while clock high
0151                                               ; for stop bit
0152   002E  0000           nop
0153   002F  0000           nop

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 4

Line   PC    Opcode

0154   0030  04C6           bcf     port_b,sclk    ; set clock low again
0155   0031  0000           nop
0156   0032  0000           nop
0157   0033  0000           nop
0158   0034  0800           retlw   0
0159                ;
0160                ;*****************************************************
0161                ;       BITOUT routine takes one bit of data in 'do' and
0162                ;       transmits it to the serial EE device
0163                ;*****************************************************
0164                BITOUT
0165   0035  0C3F           movlw   b'00111111'    ; set data,clock as outputs
0166   0036  0006           tris    port_b
0167   0037  07CA           btfss   eeprom,do      ; check for state of data bit to xmit
0168   0038  0A3B           goto    bitlow         ;
0169   0039  05E6           bsf     port_b,sdata   ; set data line high
0170   003A  0A3C           goto    clkout         ; go toggle the clock
0171                                               ;
0172   003B  04E6  bitlow    bcf     port_b,sdata   ; output a low bit
0173   003C  05C6  clkout    bsf     port_b,sclk    ; set clock line high
0174   003D  0000           nop
0175   003E  0000           nop
0176   003F  0000           nop
0177   0040  0000           nop
0178   0041  04C6           bcf     port_b,sclk    ; return clock line low
0179   0042  0800           retlw   0
0180                ;
0181                ;                  End of Subroutine
0182                ;
0183                ;*****************************************************
0184                ;       BITIN routine reads one bit of data from the
0185                ;       serial EE device and stores it in 'di'
0186                ;*****************************************************
0187                BITIN
0188   0043  05EA           bsf     eeprom,di      ; assume input bit is high
```


© 1994 Microchip Technology Inc.

```
0189   0044  0CBF          movlw   b'10111111'     ; make sdata an input line
0190   0045  0006          tris    port_b
0191   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
0192   0047  05C6          bsf     port_b,sclk     ; set clock line high
0193   0048  0000          nop                     ; just sit here a sec
0194   0049  0000          nop
0195   004A  0000          nop
0196   004B  0000          nop
0197   004C  0000          nop                     ;
0198   004D  07E6          btfss   port_b,sdata    ; read the data bit
0199   004E  04EA          bcf     eeprom,di       ; input bit was low
0200   004F  04C6          bcf     port_b,sclk     ; set clock line low
0201                                               ;
0202   0050  0800          retlw   0               ;
0203               ;
0204               ;*****************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 5

Line   PC    Opcode

0205               ;       Transmit Data Subroutine
0206               ;       This routine takes the byte of data stored in the
0207               ;       'datao' register and transmits it to the serial EE device.
0208               ;       It will then send 1 more clock to the serial EE for the
0209               ;       acknowledge bit.  If the ack bit from the part was low
0210               ;       then the transmission was successful.  If it is high, then
0211               ;       the device did not send a proper ack bit and the ack
0212               ;       fail LED will be turned on.
0213               ;*****************************************************************
0214               TX
0215   0051  0C08          movlw   .8
0216   0052  0031          movwf   count           ; set the #bits to 8
0217                       ;
0218               TXLP
0219   0053  04CA          bcf     eeprom,do       ; assume bit out is low
0220   0054  06F0          btfsc   txbuf,7         ; is bit out really low?
0221   0055  05CA          bsf     eeprom,do       ; otherwise data bit =1
0222   0056  0935          call    BITOUT          ; serial data out
0223   0057  0370          rlf     txbuf           ; rotate txbuf left
0224   0058  02F1          decfsz  count           ; 8 bits done?
0225   0059  0A53          goto    TXLP            ; no - go again
0226   005A  0943          call    BITIN           ; read ack bit
0227                       ;
0228   005B  0800          retlw   0
0229               ;                       ;
0230               ;*****************************************************************
0231               ;       Power up routine
0232               ;       This is the program entry point, which in this case simply
0233               ;       sets the port_a I/O lines and directs control to the
0234               ;       write routine.
0235               ;*****************************************************************
0236               PWRUP
0237   005C  0C00          movlw   b'00000000'
0238   005D  0005          tris    port_a          ; set port A as all output
0239   005E  0065          clrf    port_a          ; all output lines low
0240   005F  0A60          goto    WRBYTE
0241
0242               ;*****************************************************************
0243               ;       Byte Write Routine with data polling technique
0244               ;
0245               ;       This routine writes the data in "datao" to
0246               ;       8 consecutive bytes in the serial EE device starting
0247               ;       at address 00. To determine when the write cycle time
0248               ;       of the device, the 'data polling' method is used.  This
0249               ;       involves sending a start bit and control byte to the part
0250               ;       and checking for an acknowledge.  If the ack bit is low, then
0251               ;       the device is through writing, otherwise the the sequence
0252               ;       is repeated.  If no low acknowledge is found within 40
```


© 1994 Microchip Technology Inc.

```
0253                    ;       attempts (about 10 milliseconds) then the routine times
0254                    ;       out and sets the timeout LED (pin 18) high.  This program
0255                    ;       will repeat forever.

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 6

Line   PC    Opcode

0256                    ;
0257                    ;*****************************************************************
0258                    ;
0259                    WRBYTE
0260                    ;
0261   0060  0065               clrf    port_a          ; all LEDs off
0262   0061  0CA0               movlw   b'10100000'     ; set slave address and write mode
0263   0062  002F               movwf   slave
0264   0063  0CAA               movlw   b'10101010'     ; set data to write as AAh
0265   0064  002E               movwf   datao
0266   0065  0C08               movlw   .8              ; set number of bytes
0267   0066  0032               movwf   bcount          ; to write as to 8
0268   0067  0078               clrf    addr1           ; set starting high address to 00
0269   0068  006C               clrf    addr            ; set starting low address to 00
0270                                                    ;
0271   0069  090E  byte         call    BSTART          ; generate start bit
0272   006A  020F               movf    slave,w         ; move slave address
0273   006B  0030               movwf   txbuf           ; into transmit buffer
0274   006C  0951               call    TX              ; and send it
0275   006D  0218               movf    addr1,w         ; move word 1 address
0276   006E  0030               movwf   txbuf           ; into transmit buffer
0277   006F  0951               call    TX              ; and send it
0278   0070  020C               movf    addr,w          ; move word 0 address
0279   0071  0030               movwf   txbuf           ; into transmit buffer
0280   0072  0951               call    TX              ; and send it
0281   0073  020E               movf    datao,w         ; move data byte
0282   0074  0030               movwf   txbuf           ; to tranmit buffer
0283   0075  0951               call    TX              ; and transmit it
0284   0076  0923               call    BSTOP           ; generate stop bit
0285                                                    ;
0286                                                    ; now start polling for a low ack bit
0287                                                    ;
0288   0077  0C28               movlw   .40             ;
0289   0078  0037               movwf   pollcnt         ; set max number of times to poll as 40
0290   0079  090E  poll         call    BSTART          ; generate start bit
0291   007A  0CA0               movlw   b'10100000'     ; move slave address (write mode)
0292   007B  0030               movwf   txbuf           ; into transmit buffer
0293   007C  0951               call    TX              ; and send it
0294   007D  07EA               btfss   eeprom,di       ; was the ack bit low?
0295   007E  0A82               goto    exitpoll        ; yes, do another byte
0296   007F  02F7               decfsz  pollcnt         ; is poll counter down to zero?
0297   0080  0A79               goto    poll            ; no, poll again.  Other wise the part is
0298   0081  0545               bsf     port_a,timeout  ; not responding in time so set timeout
0299                                                    ; LED and continue on
0300                                                    ;
0301   0082  02AC  exitpoll     incf    addr            ; add 1 to address counter
0302   0083  02F2               decfsz  bcount          ; all 8 bytes written?
0303   0084  0A69               goto    byte            ; no, do another byte
0304   0085  0A60               goto    WRBYTE          ; yes, start over
0305                    ;
0306                    ;*****************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:54 1994  Page 7

Line   PC    Opcode

0307          0000  END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 1

Line   PC    Opcode

0001                     LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       64K Page Write Program (126 bytes)
0004                ;
0005                ;       The 24XX32/65 has a page length of 8 bytes.  This page
0006                ;       can be used to write up to 8 bytes of data into the
0007                ;       part before initiating the write cycle.  Since the
0008                ;       write cycle is timed the same for 1 byte or 8 bytes
0009                ;       it is more efficient to use the page mode when
0010                ;       consecutive addresses are being written to.
0011                ;
0012                ;       When using page mode, the control byte, upper and lower
0013                ;       addresses are sent for the first address only. After the
0014                ;       data byte for the first address is sent, the data for the
0015                ;       next consecutive address is clocked in.  This is
0016                ;       repeated as many times as needed (as long as the page
0017                ;       length is not exceeded) and then a stop bit is sent.
0018                ;       The device will still acknowledge between every byte of
0019                ;       data.  After the stop bit is sent, the part will
0020                ;       initiate the self timed write cycle.
0021                ;
0022                ;       This routine waits approximately 10mS for the write
0023                ;       cycle to complete for each page.  A more efficient
0024                ;       method of determining when the write cycle is complete
0025                ;       is called "data polling."  The data polling method is
0026                ;       explained in the program "64kdpoll.asm."  That
0027                ;       particular program uses data polling in a byte write
0028                ;       mode but it can be used in exactly the same way for
0029                ;       a page mode write.
0030                ;
0031                ;       As an option, the user can connect a LED to pin 18
0032                ;       on the PIC16C54 (use about a 1K resistor in series) as a
0033                ;       acknowledge fail indicator.  This LED will come on
0034                ;       if the device being programmed does not send a low
0035                ;       acknowledge bit at the proper times.  This can be
0036                ;       tested by simply running the program with no part
0037                ;       in the socket.
0038                ;
0039                ;       Timing is based on using the PIC16C54 in 'XT' mode
0040                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0041                ;       will be approximately 40 kHz for this setup.
0042                ;
0043                ;
0044                ;        PIC16C54 to Serial EE Connections:
0045                ;
0046                ;        PIC16C54          Serial EE
0047                ;        --------          ---------
0048                ;       Pin 12 (RB6) ->  SCLK
0049                ;       Pin 13 (RB7) ->  SDATA
0050                ;
0051                ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional)

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 2

Line   PC    Opcode

0052                ;
0053                ;
0054                ;*************************************************************
0055                ;               Register Definitions
0056                ;*************************************************************
0057          0005  port_a  equ     5h      ; port 5 (port_a) used for LEDs
0058          0006  port_b  equ     6h      ; port 6 (port b) used for data and
0059                                        ; clock lines
0060          000A  eeprom  equ     0ah     ; bit buffer
```

```
0061          000B  bycnt   equ    0bh      ; byte counter for read mode
0062          000C  addr    equ    0ch      ; word 0 address counter
0063          000D  datai   equ    0dh      ; data input register
0064          000E  datao   equ    0eh      ; data output register
0065          000F  slave   equ    0fh      ; device address (1010xxx0)
0066          0010  txbuf   equ    10h      ; transmit buffer
0067          0011  count   equ    11h      ; bit counter
0068          0012  bcount  equ    12h      ; byte counter
0069          0015  loops   equ    15h      ; delay loop counter
0070          0016  loops2  equ    16h      ; delay loop counter
0071          0017  addr1   equ    17h      ; word 1 address counter
0072                ;*************************************************************
0073                ;       Bit Definitions
0074                ;*************************************************************
0075          0007  di      equ    7        ; eeprom input bit
0076          0006  do      equ    6        ; eeprom output bit
0077          0007  sdata   equ    7        ; serial EE data line (port_b,pin 13)
0078          0006  sclk    equ    6        ; serial EE clock line (port_b,pin 12)
0079          0001  ackf    equ    1        ; acknowledge fail LED (port_a,pin 18)
0080                ;*************************************************************
0081                ;
0082                ;
0083          0000          org    01ffh    ; set reset vector
0084    01FF  0A5E          goto   PWRUP
0085          0000          org    000h     ;
0086    0000  0A5E          goto   PWRUP
0087                ;
0088                ;*************************************************************
0089                ;       DELAY ROUTINE
0090                ;       This routine takes the value in 'loops'
0091                ;       and multiplies it times 1 millisecond to
0092                ;       determine delay time.
0093                ;*************************************************************
0094                WAIT
0095                ;
0096    0001  0C6E  top     movlw   .110          ; timing adjustment variable
0097    0002  0036          movwf   loops2
0098    0003  0000  top2    nop                   ; sit and wait
0099    0004  0000          nop
0100    0005  0000          nop
0101    0006  0000          nop
0102    0007  0000          nop

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 3

Line    PC    Opcode

0103    0008  0000          nop
0104    0009  02F6          decfsz  loops2        ; inner loops complete?
0105    000A  0A03          goto    top2          ; no, go again
0106                                              ;
0107    000B  02F5          decfsz  loops         ; outer loops complete?
0108    000C  0A01          goto    top           ; no, go again
0109    000D  0800          retlw   0             ; yes, return from sub
0110                ; (Longtop)  (Optop)  (Shortop)
0111                ;*************************************************************
0112                ;       Start Bit Subroutine
0113                ;       this routine generates a start bit
0114                ;       (Low going data line while clock is high)
0115                ;*************************************************************
0116                ;
0117                BSTART
0118    000E  05E6          bsf     port_b,sdata  ; make sure data is high
0119    000F  0C3F          movlw   b'00111111'
0120    0010  0006          tris    port_b        ; set data and clock lines for output
0121    0011  04C6          bcf     port_b,sclk   ; make sure clock is low
0122    0012  0000          nop
0123    0013  05C6          bsf     port_b,sclk   ; set clock high
0124    0014  0000          nop
```


© 1994 Microchip Technology Inc.

```
0125   0015  0000          nop
0126   0016  0000          nop
0127   0017  0000          nop
0128   0018  0000          nop
0129   0019  04E6          bcf     port_b,sdata    ; data line goes low during
0130                                                ; high clock for start bit
0131   001A  0000          nop
0132   001B  0000          nop
0133   001C  0000          nop
0134   001D  0000          nop
0135   001E  0000          nop                     ; timing adjustment
0136   001F  04C6          bcf     port_b,sclk     ; start clock train
0137   0020  0000          nop
0138   0021  0000          nop
0139   0022  0800          retlw   0
0140                 ;
0141                 ;*************************************************************
0142                 ;       Stop Bit Subroutine
0143                 ;       This routine generates a stop bit
0144                 ;       (High going data line while clock is high)
0145                 ;*************************************************************
0146                 BSTOP
0147   0023  0C3F          movlw   b'00111111'     ;
0148   0024  0006          tris    port_b          ; set data/clock lines as outputs
0149   0025  04E6          bcf     port_b,sdata    ; make sure data line is low
0150   0026  0000          nop
0151   0027  0000          nop
0152   0028  0000          nop
0153   0029  05C6          bsf     port_b,sclk     ; set clock high

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 4

Line   PC    Opcode

0154   002A  0000          nop
0155   002B  0000          nop
0156   002C  0000          nop
0157   002D  05E6          bsf     port_b,sdata    ; data goes high while clock high
0158                                                ; for stop bit
0159   002E  0000          nop
0160   002F  0000          nop
0161   0030  04C6          bcf     port_b,sclk     ; set clock low again
0162   0031  0000          nop
0163   0032  0000          nop
0164   0033  0000          nop
0165   0034  0800          retlw   0
0166                 ;
0167                 ;*************************************************************
0168                 ;       BITOUT routine takes one bit of data in 'do' and
0169                 ;       transmits it to the serial EE device
0170                 ;*************************************************************
0171                 BITOUT
0172   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0173   0036  0006          tris    port_b
0174   0037  07CA          btfss   eeprom,do       ; check for state of data bit to xmit
0175   0038  0A3B          goto    bitlow          ;
0176   0039  05E6          bsf     port_b,sdata    ; set data line high
0177   003A  0A3C          goto    clkout          ; go toggle the clock
0178
0179   003B  04E6   bitlow  bcf     port_b,sdata    ; output a low bit
0180   003C  05C6   clkout  bsf     port_b,sclk     ; set clock line high
0181   003D  0000          nop
0182   003E  0000          nop
0183   003F  0000          nop
0184   0040  0000          nop
0185   0041  04C6          bcf     port_b,sclk     ; return clock line low
0186   0042  0800          retlw   0
0187                 ;
0188                 ;*************************************************************
```

8

© 1994 Microchip Technology Inc.

```
0189                    ;       BITIN routine reads one bit of data from the
0190                    ;       serial EE device and stores it in 'di'
0191                    ;*****************************************************************
0192                    BITIN
0193    0043  05EA              bsf     eeprom,di       ; assume input bit is high
0194    0044  0CBF              movlw   b'10111111'     ; make sdata an input line
0195    0045  0006              tris    port_b
0196    0046  05E6              bsf     port_b,sdata    ; set sdata line for input
0197    0047  05C6              bsf     port_b,sclk     ; set clock line high
0198    0048  0000              nop                     ; just sit here a sec
0199    0049  0000              nop
0200    004A  0000              nop
0201    004B  0000              nop
0202    004C  0000              nop                     ;
0203    004D  07E6              btfss   port_b,sdata    ; read the data bit
0204    004E  04EA              bcf     eeprom,di       ; input bit was low

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 5

Line    PC    Opcode

0205    004F  04C6              bcf     port_b,sclk     ; set clock line low
0206                                                    ;
0207    0050  0800              retlw   0               ;
0208                    ;
0209                    ;*****************************************************************
0210                    ;       Transmit Data Subroutine
0211                    ;       This routine takes the byte of data stored in the
0212                    ;       'datao' register and transmits it to the serial EE device.
0213                    ;       It will then send 1 more clock to the serial EE for the
0214                    ;       acknowledge bit.  If the ack bit from the part was low
0215                    ;       then the transmission was successful.  If it is high, then
0216                    ;       the device did not send a proper ack bit and the ack
0217                    ;       fail LED will be turned on.
0218                    ;*****************************************************************
0219                    TX
0220    0051  0C08              movlw   .8
0221    0052  0031              movwf   count           ; set the #bits to 8
0222                            ;
0223                    TXLP
0224    0053  04CA              bcf     eeprom,do       ; assume bit out is low
0225    0054  06F0              btfsc   txbuf,7         ; is bit out really low?
0226    0055  05CA              bsf     eeprom,do       ; otherwise data bit =1
0227    0056  0935              call    BITOUT          ; serial data out
0228    0057  0370              rlf     txbuf           ; rotate txbuf left
0229    0058  02F1              decfsz  count           ; 8 bits done?
0230    0059  0A53              goto    TXLP            ; no - go again
0231    005A  0943              call    BITIN           ; read ack bit
0232    005B  06EA              btfsc   eeprom,di       ; check ack bit
0233    005C  0525              bsf     port_a,ackf     ; set acknowledge fail LED if the
0234                            ;
0235    005D  0800              retlw   0
0236                    ;
0237                    ;*****************************************************************
0238                    ;       Power up routine
0239                    ;       This is the program entry point, which in this case simply
0240                    ;       sets the port_a I/O lines and directs control to the
0241                    ;       byte write routine.
0242                    ;*****************************************************************
0243                    PWRUP
0244    005E  0C00              movlw   b'00000000'
0245    005F  0005              tris    port_a          ; set port A as all output
0246    0060  0065              clrf    port_a          ; all output lines low
0247    0061  0A62              goto    pwrite          ; go do the page write
0248
0249                    ;*****************************************************************
0250                    ;       Page Write Routine
0251                    ;       This routine writes the data in "datao" to 8 consecutive
0252                    ;       bytes using page write mode starting at address 00.
```


© 1994 Microchip Technology Inc.

```
0253                ;       This routine waits 10mS after every page to give the device
0254                ;       time to do the write.  This program repeats forever.
0255                ;*************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:48 1994  Page 6

Line   PC    Opcode

0256                ;
0257                pwrite
0258                ;
0259   0062  0065           clrf    port_a          ; clear all LEDs
0260   0063  0CA0           movlw   b'10100000'     ; set slave address and write mode
0261   0064  002F           movwf   slave
0262   0065  0C55           movlw   b'01010101'     ; set data to write as 55h
0263   0066  002E           movwf   datao
0264                ;
0265   0067  0C08           movlw   .8              ; set number of bytes
0266   0068  0032           movwf   bcount          ; to write as to 8
0267   0069  0077           clrf    addr1           ; set high address byte to 00
0268   006A  006C           clrf    addr            ; set low address byte to 00
0269                                                ;
0270   006B  090E           call    BSTART          ; generate start bit
0271   006C  020F           movf    slave,w         ; move slave address
0272   006D  0030           movwf   txbuf           ; into transmit buffer
0273   006E  0951           call    TX              ; and send it
0274   006F  0217           movf    addr1,w         ; move word 1 address
0275   0070  0030           movwf   txbuf           ; into transmit buffer
0276   0071  0951           call    TX              ; and send it
0277   0072  020C           movf    addr,w          ; move word 0 address
0278   0073  0030           movwf   txbuf           ; into transmit buffer
0279   0074  0951           call    TX              ; and send it
0280
0281   0075  020E  byte      movf    datao,w         ; move data byte
0282   0076  0030           movwf   txbuf           ; to transmit buffer
0283   0077  0951           call    TX              ; and transmit it
0284   0078  02F2           decfsz  bcount          ; all 8 bytes written?
0285   0079  0A75           goto    byte            ; no, do another byte
0286                                                ;
0287   007A  0923           call    BSTOP           ; generate stop bit
0288   007B  0C0A           movlw   .10
0289   007C  0035           movwf   loops           ; set delay time to give
0290   007D  0901           call    WAIT            ; 10 ms wait after every byte
0291   007E  0A62           goto    pwrite          ; start over
0292                ;
0293                ;
0294          0000  END
```

8

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       64K Sequential Read Program (142 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 24XX32/65 Serial EE device
0007                ;       and perform a sequential read operation. This program
0008                ;       will read 8 consecutive addresses in the 'sequential
0009                ;       read' mode.  This entails setting the address pointer
0010                ;       for the first address only and then clocking out as many
0011                ;       bytes of data as needed.  The device will automatically
0012                ;       increment the address.
0013                ;
0014                ;       Timing is based on using the PIC16C54 in 'XT' mode
0015                ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0016                ;       will be approximately 60 kHz for this setup.
0017                ;
0018                ;       As an option, the user may connect a LED to pin 18
0019                ;       on the PIC16C54 (use about a 1K resistor in series) as an
0020                ;       acknowledge fail indicator.  This LED will come on if
0021                ;       the serial EE fails to acknowledge correctly.
0022                ;
0023                ;        PIC16C54 to Serial EE Connections:
0024                ;
0025                ;        PIC16C54          Serial EE
0026                ;        --------          ---------
0027                ;       Pin 12 (RB6) -->  SCLK
0028                ;       Pin 13 (RB7) -->  SDATA
0029                ;
0030                ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
0031                ;
0032                ;
0033                ;*************************************************************
0034                ;               Register Definitions
0035                ;*************************************************************
0036         0003   status  equ    03h     ; status register
0037         0005   port_a  equ    05h     ; port 5 (port a) used for LED display
0038         0006   port_b  equ    06h     ; port 6 (port b) used for data and
0039                                        ; clock lines
0040         000A   eeprom  equ    0ah     ; bit buffer
0041         000B   bycnt   equ    0bh     ; byte counter for read mode
0042         000C   addr    equ    0ch     ; word 0 address counter
0043         000D   datai   equ    0dh     ; data input register
0044         000E   datao   equ    0eh     ; data output register
0045         000F   slave   equ    0fh     ; device address 1010xxx0)
0046         0010   txbuf   equ    10h     ; transmit buffer
0047         0011   count   equ    11h     ; bit counter
0048         0012   bcount  equ    12h     ; byte counter
0049         0015   loops   equ    15h     ; delay loop counter
0050         0016   loops2  equ    16h     ; delay loop counter
0051         0017   addr1   equ    17h     ; word 1 address counter

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 2

Line   PC    Opcode

0052                ;*************************************************************
0053                ;               Bit Definitions
0054                ;*************************************************************
0055         0007   di      equ    7       ; eeprom input bit
0056         0006   do      equ    6       ; eeprom output bit
0057         0007   sdata   equ    7       ; serial EE data line (port_b,pin 13)
0058         0006   sclk    equ    6       ; serial EE clock line (port_b,pin 12)
0059         0001   ackf    equ    1       ; acknowledge fail LED (port_a)
0060                ;*************************************************************
```


© 1994 Microchip Technology Inc.

```
0061                    ;
0062    0000            org     01ffh   ; set reset vector
0063    01FF    0A69            goto    PWRUP
0064    0000            org     000h    ;
0065    0000    0A69            goto    PWRUP
0066                    ;
0067                    ;*************************************************************
0068                    ;       DELAY ROUTINE
0069                    ;       This routine takes the value in 'loops'
0070                    ;       and multiplies it times 1 millisecond to
0071                    ;       determine delay time.
0072                    ;*************************************************************
0073                    WAIT
0074                    ;
0075    0001    0C6E    top     movlw   .110            ; timing adjustment variable
0076    0002    0036            movwf   loops2
0077    0003    0000    top2    nop                     ; sit and wait
0078    0004    0000            nop
0079    0005    0000            nop
0080    0006    0000            nop
0081    0007    0000            nop
0082    0008    0000            nop
0083    0009    02F6            decfsz  loops2          ; inner loops complete?
0084    000A    0A03            goto    top2            ; no, go again
0085                                                    ;
0086    000B    02F5            decfsz  loops           ; outer loops complete?
0087    000C    0A01            goto    top             ; no, go again
0088    000D    0800            retlw   0               ; yes, return from sub
0089                    ;
0090                    ;*************************************************************
0091                    ;       Start Bit Subroutine
0092                    ;       this routine generates a start bit
0093                    ;       (Low going data line while clock is high)
0094                    ;*************************************************************
0095                    ;
0096                    BSTART
0097    000E    05E6            bsf     port_b,sdata    ; make sure data is high
0098    000F    0C3F            movlw   b'00111111'
0099    0010    0006            tris    port_b          ; set data and clock lines for output
0100    0011    04C6            bcf     port_b,sclk     ; make sure clock is low
0101    0012    0000            nop
0102    0013    05C6            bsf     port_b,sclk     ; set clock high

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 3

Line    PC      Opcode

0103    0014    0000            nop
0104    0015    0000            nop
0105    0016    0000            nop
0106    0017    0000            nop
0107    0018    0000            nop
0108    0019    04E6            bcf     port_b,sdata    ; data line goes low during
0109                                                    ; high clock for start bit
0110    001A    0000            nop
0111    001B    0000            nop
0112    001C    0000            nop
0113    001D    0000            nop
0114    001E    0000            nop                     ; timing adjustment
0115    001F    04C6            bcf     port_b,sclk     ; start clock train
0116    0020    0000            nop
0117    0021    0000            nop
0118    0022    0800            retlw   0
0119                            ;
0120                    ;*************************************************************
0121                    ;       Stop Bit Subroutine
0122                    ;       This routine generates a stop bit
0123                    ;       (High going data line while clock is high)
0124                    ;*************************************************************
```

© 1994 Microchip Technology Inc.

```
0125                    BSTOP
0126    0023    04E6            bcf     port_b,sdata    ; make sure data line is low
0127    0024    0C3F            movlw   b'00111111'     ;
0128    0025    0006            tris    port_b          ; set data/clock lines as outputs
0129    0026    04E6            bcf     port_b,sdata    ; make sure data line is low
0130    0027    0000            nop
0131    0028    0000            nop
0132    0029    0000            nop
0133    002A    05C6            bsf     port_b,sclk     ; set clock high
0134    002B    0000            nop
0135    002C    0000            nop
0136    002D    0000            nop
0137    002E    05E6            bsf     port_b,sdata    ; data goes high while clock high
0138                                                    ; for stop bit
0139    002F    0000            nop
0140    0030    0000            nop
0141    0031    04C6            bcf     port_b,sclk     ; set clock low again
0142    0032    0000            nop
0143    0033    0000            nop
0144    0034    0000            nop
0145    0035    0800            retlw   0
0146                    ;
0147                    ;*****************************************************
0148                    ;       BITOUT routine takes the bit of data in 'do' and
0149                    ;       transmits it to the serial EE device
0150                    ;*****************************************************
0151                    BITOUT
0152    0036    0C3F            movlw   b'00111111'     ; set data,clock as outputs
0153    0037    0006            tris    port_b

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 4

Line    PC      Opcode

0154    0038    07CA            btfss   eeprom,do       ; check for state of data bit to xmit
0155    0039    0A3C            goto    bitlow          ; low? go set data line low
0156    003A    05E6            bsf     port_b,sdata    ; high? set data line high
0157    003B    0A3D            goto    clkout          ; go toggle the clock
0158
0159    003C    04E6    bitlow  bcf     port_b,sdata    ; output a low bit
0160    003D    05C6    clkout  bsf     port_b,sclk     ; set clock line high
0161    003E    0000            nop
0162    003F    0000            nop
0163    0040    0000            nop
0164    0041    0000            nop
0165    0042    04C6            bcf     port_b,sclk     ; return clock line low
0166    0043    0800            retlw   0
0167                    ;
0168                    ;*****************************************************
0169                    ;       BITIN routine reads one bit of data from the
0170                    ;       serial EE device and stores it in the bit 'di'
0171                    ;*****************************************************
0172                    BITIN
0173    0044    05EA            bsf     eeprom,di       ; assume input bit is high
0174    0045    0CBF            movlw   b'10111111'     ; make sdata an input line
0175    0046    0006            tris    port_b
0176    0047    05C6            bsf     port_b,sclk     ; set clock line high
0177    0048    0000            nop                     ; just sit here a sec
0178    0049    0000            nop
0179    004A    0000            nop
0180    004B    0000            nop
0181    004C    0000            nop                     ;
0182    004D    07E6            btfss   port_b,sdata    ; read the data bit
0183    004E    04EA            bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0184    004F    04C6            bcf     port_b,sclk     ; set clock line low
0185                                                    ;
0186    0050    0800            retlw   0               ;
0187                    ;
0188                    ;*****************************************************
```


© 1994 Microchip Technology Inc.

```
0189                    ;       Transmit Data Subroutine
0190                    ;       This routine takes the byte of data stored in the
0191                    ;       'datao' register and transmits it to the serial EE device.
0192                    ;       It will then send 1 more clock to the serial EE for the
0193                    ;       acknowledge bit.  If the ack bit from the part was low
0194                    ;       then the transmission was successful.  If it is high, then
0195                    ;       the device did not send a proper ack bit and the ack
0196                    ;       fail LED will be turned on.
0197                    ;*****************************************************************
0198                    TX
0199    0051    0C08            movlw   .8
0200    0052    0031            movwf   count           ; set the #bits to 8
0201                            ;
0202                    TXLP
0203    0053    04CA            bcf     eeprom,do       ; assume bit out is low
0204    0054    06F0            btfsc   txbuf,7         ; is bit out really low?

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 5

Line    PC      Opcode

0205    0055    05CA            bsf     eeprom,do       ; no, set it high
0206    0056    0936            call    BITOUT          ; send the bit to serial EE
0207    0057    0370            rlf     txbuf           ; rotate txbuf left
0208    0058    02F1            decfsz  count           ; 8 bits done?
0209    0059    0A53            goto    TXLP            ; no - go again
0210    005A    0944            call    BITIN           ; read ack bit
0211    005B    06EA            btfsc   eeprom,di       ; check ack bit
0212    005C    0525            bsf     port_a,ackf     ; set acknowledge fail LED if the
0213                                                    ; device did not pull data low
0214                            ;
0215    005D    0800            retlw   0
0216                    ;
0217                    ;*****************************************************************
0218                    ;       Receive data routine
0219                    ;       This routine reads one byte of data from the part
0220                    ;       into the 'datai' register.  It then sends a high
0221                    ;       ack bit to indicate that no more data is to be read
0222                    ;*****************************************************************
0223                    RX
0224    005E    006D            clrf    datai           ; clear input buffer
0225    005F    0C08            movlw   .8              ; set # bits to 8
0226    0060    0031            movwf   count
0227    0061    0403            bcf     status,0        ; make sure carry bit is low
0228    0062    036D    RXLP    rlf     datai           ; rotate datai 1 bit left
0229    0063    0944            call    BITIN           ; read a bit
0230    0064    06EA            btfsc   eeprom,di
0231    0065    050D            bsf     datai,0         ; set bit 0 if necessary
0232    0066    02F1            decfsz  count           ; 8 bits done?
0233    0067    0A62            goto    RXLP            ; no, do another
0234    0068    0800            retlw   0
0235                    ;
0236                    ;*****************************************************************
0237                    ;       Power up routine
0238                    ;       This is the program entry point.  I/O line status is
0239                    ;       set for port A here.
0240                    ;*****************************************************************
0241                    PWRUP
0242    0069    0C00            movlw   b'00000000'
0243    006A    0005            tris    port_a          ; set port A as all output
0244    006B    0065            clrf    port_a          ; all output lines low
0245    006C    0A6D            goto    READ
0246                    ;
0247                    ;*****************************************************************
0248                    ;       READ (read routine)
0249                    ;       This routine reads 8 consecutive addresses of the
0250                    ;       serial EE device starting at address 00 in the
0251                    ;       sequential read mode. Reading in this mode is more
0252                    ;       efficient than the random read mode as the control byte
```

8

© 1994 Microchip Technology Inc.

```
0253                    ;       and address have to be sent only once at the beginning
0254                    ;       of the sequence.  As many consecutive addresses as
0255                    ;       needed can then be read from the part until a stop bit is

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:33 1994  Page 6

Line   PC    Opcode

0256                    ;       sent.  In the read mode, the 16C54 must send the acknowledge
0257                    ;       bit after every 8 data bits from the device.  When the
0258                    ;       last byte needed has been read, then the controller will
0259                    ;       send a high acknowledge bit and then a stop bit to halt
0260                    ;       transmission from the device.
0261                    ;*************************************************************
0262                    READ
0263                    ;
0264   006D  0425               bcf     port_a,ackf ; clear the ack fail LED if on
0265   006E  0C08               movlw   .8
0266   006F  0032               movwf   bcount      ; set number of bytes to read as 8
0267   0070  0CA0               movlw   b'10100000' ; set slave address and write mode
0268   0071  002F               movwf   slave
0269   0072  0077               clrf    addr1       ; set starting high address to 00
0270   0073  006C               clrf    addr        ; set starting low address to 00
0271                                                 ;
0272   0074  090E               call    BSTART      ; generate start bit
0273   0075  020F               movf    slave,w     ; get slave address
0274   0076  0030               movwf   txbuf       ; into transmit buffer
0275   0077  0951               call    TX          ; and send it
0276   0078  0217               movf    addr1,w     ; get word 1 address
0277   0079  0030               movwf   txbuf       ; into transmit buffer
0278   007A  0951               call    TX          ; and send it
0279   007B  020C               movf    addr,w      ; get word 0 address
0280   007C  0030               movwf   txbuf       ; into transmit buffer
0281   007D  0951               call    TX          ; and send it
0282   007E  090E               call    BSTART      ; generate start bit
0283   007F  0CA1               movlw   b'10100001' ; get slave address and read mode
0284   0080  0030               movwf   txbuf       ; into transmit buffer
0285   0081  0951               call    TX          ; and transmit it
0286                                                 ;
0287   0082  095E  rbyte        call    RX          ; read 1 byte from device
0288   0083  02F2               decfsz  bcount      ; are all 8 bytes read?
0289   0084  0A8C               goto    lowack      ; no, send low ack and do another
0290   0085  05CA               bsf     eeprom,do   ; yes, send high ack bit
0291   0086  0936               call    BITOUT      ; to stop transmission
0292   0087  0923               call    BSTOP       ; and send a stop bit
0293
0294   0088  0CFF               movlw   .255        ; long delay for scope
0295   0089  0035               movwf   loops       ; trigger purposes only
0296   008A  0901               call    wait
0297   008B  0A6D               goto    READ        ; start all over
0298
0299   008C  04CA  lowack       bcf     eeprom,do   ; send low ack bit
0300   008D  0936               call    BITOUT      ; to continue transmission
0301   008E  0A82               goto    rbyte       ; and read another byte
0302                    ;
0303                    ;
0304          0000      END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*************************************************************
0003                ;       Program to set the high endurance block on the 64K
0004                ;       device (24XX65). This routine does not apply to the 24XX32.(122 bytes)
0005                ;
0006                ;       The 24LC65 is a 64K device, 4K of which is error
0007                ;       corrected to increase the endurance of that portion
0008                ;       of the array.  The location of this high endurance
0009                ;       'block' can be set by the user.  When the device
0010                ;       comes from the factory, the high endurance block
0011                ;       will be set as the uppermost block in the array.
0012                ;
0013                ;       This program sets the high endurance block as the
0014                ;       first block in the array (address range 00h to 1FFh).
0015                ;
0016                ;       The high endurance block is set by sending the following
0017                ;       sequence to the device:
0018                ;
0019                ;       SB 10100000 1XXAAAAX XXXXXXXX 00XX0000 STB
0020                ;
0021                ;       Where SB=start bit
0022                ;              1=data high
0023                ;              0=data low
0024                ;              X=don't care
0025                ;             AAAA=4 bit high endurance block address
0026                ;              STB=stop bit
0027                ;
0028                ;       As an option, the user may connect a LED to pin 18
0029                ;       on the PIC16C54 (use about a 1K resistor in series) as an
0030                ;       acknowledge fail indicator.  This LED will come on if
0031                ;       the serial EE fails to acknowledge correctly.
0032                ;
0033                ;       Timing is based on using the PIC16C54 in 'XT' mode
0034                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0035                ;       will be approximately 60 kHz for this setup.
0036                ;
0037                ;        PIC16C54 to Serial EE Connections:
0038                ;
0039                ;          PIC16C54        Serial EE
0040                ;          --------        ---------
0041                ;       Pin 12 (RB6) ->  SCLK
0042                ;       Pin 13 (RB7) ->  SDATA
0043                ;
0044                ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional);
0045                ;*************************************************************
0046                ;             Register Definitions
0047                ;*************************************************************
0048         0005   port_a  equ     5h      ; port 5 (port_a) used for LEDs
0049         0006   port_b  equ     6h      ; port 6 (port b) used for data and
0050                                        ; clock lines
0051         000A   eeprom  equ     0ah     ; bit buffer

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 2

Line   PC    Opcode

0052         000B   bycnt   equ     0bh     ; byte counter for read mode
0053         000C   addr    equ     0ch     ; word 0 address counter
0054         000D   datai   equ     0dh     ; data input register
0055         000E   datao   equ     0eh     ; data output register
0056         000F   slave   equ     0fh     ; device address (1010xxx0)
0057         0010   txbuf   equ     10h     ; transmit buffer
0058         0011   count   equ     11h     ; bit counter
0059         0012   bcount  equ     12h     ; byte counter
0060         0015   loops   equ     15h     ; delay loop counter
```

8

© 1994 Microchip Technology Inc.

```
0061          0016  loops2  equ    16h     ; delay loop counter
0062          0017  addr1   equ    17h     ; word 1 address counter
0063          0018  he_blk  equ    18h     ; high endurance block address
0064                ;*************************************************************
0065                ;           Bit Definitions
0066                ;*************************************************************
0067          0007  di       equ    7       ; eeprom input bit
0068          0006  do       equ    6       ; eeprom output bit
0069          0007  sdata    equ    7       ; serial EE data line (port_b,pin 13)
0070          0006  sclk     equ    6       ; serial EE clock line (port_b,pin 12)
0071          0001  ackf     equ    1       ; acknowledge fail LED (port_a,pin 18)
0072                ;*************************************************************
0073                ;
0074                ;
0075          0000           org    01ffh   ; set reset vector
0076    01FF  0A5E           goto   PWRUP
0077          0000           org    000h    ;
0078    0000  0A5E           goto   PWRUP
0079                ;
0080                ;*************************************************************
0081                ;       DELAY ROUTINE
0082                ;       This routine takes the value in 'loops'
0083                ;       and multiplies it times 1 millisecond to
0084                ;       determine delay time.
0085                ;*************************************************************
0086                WAIT
0087                ;
0088    0001  0C6E  top      movlw   .110            ; timing adjustment variable
0089    0002  0036           movwf   loops2
0090    0003  0000  top2     nop                     ; sit and wait
0091    0004  0000           nop
0092    0005  0000           nop
0093    0006  0000           nop
0094    0007  0000           nop
0095    0008  0000           nop
0096    0009  02F6           decfsz  loops2          ; inner loops complete?
0097    000A  0A03           goto    top2            ; no, go again
0098                                                 ;
0099    000B  02F5           decfsz  loops           ; outer loops complete?
0100    000C  0A01           goto    top             ; no, go again
0101    000D  0800           retlw   0               ; yes, return from sub
0102                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 3

Line    PC    Opcode

0103                ;*************************************************************
0104                ;       Start Bit Subroutine
0105                ;       this routine generates a start bit
0106                ;       (Low going data line while clock is high)
0107                ;*************************************************************
0108                ;
0109                BSTART
0110    000E  05E6           bsf     port_b,sdata    ; make sure data is high
0111    000F  0C3F           movlw   b'00111111'
0112    0010  0006           tris    port_b          ; set data and clock lines for output
0113    0011  04C6           bcf     port_b,sclk     ; make sure clock is low
0114    0012  0000           nop
0115    0013  05C6           bsf     port_b,sclk     ; set clock high
0116    0014  0000           nop
0117    0015  0000           nop
0118    0016  0000           nop
0119    0017  0000           nop
0120    0018  0000           nop
0121    0019  04E6           bcf     port_b,sdata    ; data line goes low during
0122                                                 ; high clock for start bit
0123    001A  0000           nop
0124    001B  0000           nop
```


© 1994 Microchip Technology Inc.

```
0125   001C  0000          nop
0126   001D  0000          nop
0127   001E  0000          nop                     ; timing adjustment
0128   001F  04C6          bcf     port_b,sclk     ; start clock train
0129   0020  0000          nop
0130   0021  0000          nop
0131   0022  0800          retlw   0
0132                ;
0133                ;*****************************************************************
0134                ;       Stop Bit Subroutine
0135                ;       This routine generates a stop bit
0136                ;       (High going data line while clock is high)
0137                ;*****************************************************************
0138                BSTOP
0139   0023  0C3F          movlw   b'00111111'     ;
0140   0024  0006          tris    port_b          ; set data/clock lines as outputs
0141   0025  04E6          bcf     port_b,sdata    ; make sure data line is low
0142   0026  0000          nop
0143   0027  0000          nop
0144   0028  0000          nop
0145   0029  05C6          bsf     port_b,sclk     ; set clock high
0146   002A  0000          nop
0147   002B  0000          nop
0148   002C  0000          nop
0149   002D  05E6          bsf     port_b,sdata    ; data goes high while clock high
0150                                                ; for stop bit
0151   002E  0000          nop
0152   002F  0000          nop
0153   0030  04C6          bcf     port_b,sclk     ; set clock low again

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 4

Line   PC    Opcode

0154   0031  0000          nop
0155   0032  0000          nop
0156   0033  0000          nop
0157   0034  0800          retlw   0
0158                ;
0159                ;*****************************************************************
0160                ;       BITOUT routine takes one bit of data in 'do' and
0161                ;       transmits it to the serial EE device
0162                ;*****************************************************************
0163                BITOUT
0164   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0165   0036  0006          tris    port_b
0166   0037  07CA          btfss   eeprom,do       ; check for state of data bit to xmit
0167   0038  0A3B          goto    bitlow          ;
0168   0039  05E6          bsf     port_b,sdata    ; set data line high
0169   003A  0A3C          goto    clkout          ; go toggle the clock
0170
0171   003B  04E6  bitlow  bcf     port_b,sdata    ; output a low bit
0172   003C  05C6  clkout  bsf     port_b,sclk     ; set clock line high
0173   003D  0000          nop
0174   003E  0000          nop
0175   003F  0000          nop
0176   0040  0000          nop
0177   0041  04C6          bcf     port_b,sclk     ; return clock line low
0178   0042  0800          retlw   0
0179                ;
0180                ;*****************************************************************
0181                ;       BITIN routine reads one bit of data from the
0182                ;       serial EE device and stores it in 'di'
0183                ;*****************************************************************
0184                BITIN
0185   0043  05EA          bsf     eeprom,di       ; assume input bit is high
0186   0044  0CBF          movlw   b'10111111'     ; make sdata an input line
0187   0045  0006          tris    port_b
0188   0046  05E6          bsf     port_b,sdata    ; set sdata line for input
```

8

© 1994 Microchip Technology Inc.

```
0189   0047  05C6          bsf     port_b,sclk     ; set clock line high
0190   0048  0000          nop                     ; just sit here a sec
0191   0049  0000          nop
0192   004A  0000          nop
0193   004B  0000          nop
0194   004C  0000          nop                     ;
0195   004D  07E6          btfss   port_b,sdata    ; read the data bit
0196   004E  04EA          bcf     eeprom,di       ; input bit was low
0197   004F  04C6          bcf     port_b,sclk     ; set clock line low
0198                                               ;
0199   0050  0800          retlw   0               ;
0200               ;
0201               ;*****************************************************************
0202               ;       Transmit Data Subroutine
0203               ;       This routine takes the byte of data stored in the
0204               ;       'datao' register and transmits it to the serial EE device.

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 5

Line   PC    Opcode

0205               ;       It will then send 1 more clock to the serial EE for the
0206               ;       acknowledge bit.  If the ack bit from the part was low
0207               ;       then the transmission was successful.  If it is high, then
0208               ;       the device did not send a proper ack bit and the ack
0209               ;       fail LED will be turned on.
0210               ;*****************************************************************
0211               TX
0212   0051  0C08          movlw   .8
0213   0052  0031          movwf   count           ; set the #bits to 8
0214                       ;
0215               TXLP
0216   0053  04CA          bcf     eeprom,do       ; assume bit out is low
0217   0054  06F0          btfsc   txbuf,7         ; is bit out really low?
0218   0055  05CA          bsf     eeprom,do       ; otherwise data bit =1
0219   0056  0935          call    BITOUT          ; serial data out
0220   0057  0370          rlf     txbuf           ; rotate txbuf left
0221   0058  02F1          decfsz  count           ; 8 bits done?
0222   0059  0A53          goto    TXLP            ; no - go again
0223   005A  0943          call    BITIN           ; read ack bit
0224   005B  06EA          btfsc   eeprom,di       ; check ack bit
0225   005C  0525          bsf     port_a,ackf     ; set acknowledge fail LED if the
0226                       ;
0227   005D  0800          retlw   0
0228               ;
0229               ;*****************************************************************
0230               ;       Power up routine
0231               ;       This is the program entry point, which in this case simply
0232               ;       sets the port_a I/O lines and directs control to the
0233               ;       byte write routine.
0234               ;*****************************************************************
0235               PWRUP
0236   005E  0C00          movlw   b'00000000'
0237   005F  0005          tris    port_a          ; set port A as all output
0238   0060  0065          clrf    port_a          ; all output lines low
0239   0061  0A62          goto    setheblk        ; set go set the high endurance block
0240
0241               ;*****************************************************************
0242               ;       Set High Endurance Block Routine
0243               ;       This routine sets the high endurance block to the first
0244               ;       block in the array and then delays about a half second.  This
0245               ;       routine will repeat forever.
0246               ;*****************************************************************
0247               ;
0248               setheblk
0249               ;
0250   0062  0065          clrf    port_a          ; clear all LEDs
0251   0063  002F          movwf   slave
0252   0064  0C00          movlw   .0
```


© 1994 Microchip Technology Inc.

```
0253   0065  0038          movwf   he_blk          ; set the endurance block as first
0254                                               ; block in array (block 0)
0255   0066  0378          rlf     he_blk          ; rotate starting block left 1 bit
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:10 1994  Page 6

Line   PC    Opcode

0256                                               ; to arrange it correctly
0257   0067  05F8          bsf     he_blk,7        ; set msb bit in block address byte
0258   0068  090E          call    BSTART          ; generate start bit
0259   0069  0CA0          movlw   b'10100000'     ; get slave address and write mode
0260   006A  0030          movwf   txbuf           ; into transmit buffer
0261   006B  0951          call    TX              ; and send it
0262   006C  0218          movf    he_blk,w        ; get the block address byte
0263   006D  0030          movwf   txbuf           ; into transmit buffer
0264   006E  0951          call    TX              ; and send it
0265   006F  0070          clrf    txbuf           ; clear buffer
0266   0070  0951          call    TX              ; send 8 don't care bits
0267   0071  0070          clrf    txbuf
0268   0072  0951          call    TX              ; send config byte(all 0's)
0269   0073  0923          call    BSTOP           ; send stop bit
0270
0271
0272   0074  0CFF          movlw   .255            ; long delay
0273   0075  0035          movwf   loops
0274   0076  0901          call    wait
0275   0077  0CFF          movlw   .255
0276   0078  0035          movwf   loops
0277   0079  0901          call    wait
0278
0279   007A  0A62          goto    setheblk        ; go again
0280               ;
0281               ;
0282         0000  END
```

8

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 1
Line   PC    Opcode

0001                                                        LIST   P=16C54,c=132   0002
;************************************************************ 0003          ;       Program to
set the security option (write protect)
0004                ;       on the 64K device (24LC65).   (125 bytes)
0005                ;
0006                ;       The security option (or write protect option) allows
0007                ;       the user to write protect any number of consecutive
0008                ;       4K blocks in the device.
0009                ;
0010                ;       This program sets the security option to protect the
0011                ;       lower 4 blocks (2K bytes) in the array.
0012                ;
0013                ;       **********  CAUTIONS TO THE USER !!!! **********
0014                ;
0015                ;       1) THE SECURITY OPTION CAN ONLY BE SET ONCE!!
0016                ;       2) THE HIGH ENDURANCE BLOCK CANNOT BE CHANGED
0017                ;          AFTER THE SECURITY OPTION IS SET.
0018                ;       3) THE HIGH ENDURANCE BLOCK CANNOT BE PROTECTED.
0019                ;
0020                ;       The security option is set by sending the following
0021                ;       sequence to the device:
0022                ;
0023                ;       SB 10100000 1XXAAAAX XXXXXXXX 10XXNNNN STB
0024                ;
0025                ;       Where SB=start bit
0026                ;              1=data high
0027                ;              0=data low
0028                ;              X=don't care
0029                ;             AAAA=4 bit number to indicate first secure block (0-15)
0030                ;             NNNN=4 bit number to indicate how many secure blocks(1-15)
0031                ;              STB=stop bit
0032                ;
0033                ;       As an option, the user may connect a LED to pin 18
0034                ;       on the PIC 16C54 (use about a 1K resistor in series) as an
0035                ;       acknowledge fail indicator.  This LED will come on if
0036                ;       the serial EE fails to acknowledge correctly.
0037                ;
0038                ;       Timing is based on using the PIC 16C54 in 'XT' mode
0039                ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0040                ;       will be approximately 60 Khz for this setup.
0041                ;
0042                ;        PIC 16C54 to Serial EE Connections:
0043                ;
0044                ;        PIC 16C54        Serial EE
0045                ;       ------    ------
0046                ;       Pin 12 (RB6) ->  SCLK
0047                ;       Pin 13 (RB7) ->  SDATA
0048                ;
0049                ;       PIN 18 (RA1) ->  Acknowledge fail LED  (Optional);
0050                ;************************************************************
0051                ;           Register Definitions
 16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 2
Line   PC    Opcode

0052                ;************************************************************ 0053          0005
port_a      equ     5h      ; port 5 (port_a) used for LEDs
0054          0006  port_b      equ     6h      ; port 6 (port b) used for data and
0055                                            ; clock lines
0056          000A  eeprom      equ     0ah     ; bit buffer
0057          000B  bycnt       equ     0bh     ; byte counter for read mode
0058          000C  addr        equ     0ch     ; word 0 address counter
0059          000D  datai       equ     0dh     ; data input register
0060          000E  datao       equ     0eh     ; data output register
0061          000F  slave       equ     0fh     ; device address (1010xxx0)
0062          0010  txbuf       equ     10h     ; transmit buffer
0063          0011  count       equ     11h     ; bit counter
```


© 1994 Microchip Technology Inc.

```
0055                                        ; clock lines
0056          000A  eeprom     equ    0ah     ; bit buffer
0057          000B  bycnt      equ    0bh     ; byte counter for read mode
0058          000C  addr       equ    0ch     ; word 0 address counter
0059          000D  datai      equ    0dh     ; data input register
0060          000E  datao      equ    0eh     ; data output register
0061          000F  slave      equ    0fh     ; device address (1010xxx0)
0062          0010  txbuf      equ    10h     ; transmit buffer
0063          0011  count      equ    11h     ; bit counter
0064          0012  bcount     equ    12h     ; byte counter
0065          0015  loops      equ    15h     ; delay loop counter
0066          0016  loops2     equ    16h     ; delay loop counter
0067          0017  addr1      equ    17h     ; word 1 address counter
0068          0018  strt_blk   equ    18h     ; starting block number for secure portion
0069          0019  sec_blks   equ    19h     ; number of secure blocks
0070                ;*************************************************************
0071                ;           Bit Definitions
0072                ;*************************************************************
0073          0007  di         equ    7       ; eeprom input bit
0074          0006  do         equ    6       ; eeprom output bit
0075          0007  sdata      equ    7       ; serial EE data line (port_b,pin 13)
0076          0006  sclk       equ    6       ; serail EE clock line (port_b,pin 12)
0077          0001  ackf       equ    1       ; acknowledge fail LED (port_a,pin 18)
0078                ;*************************************************************
0079              ; 0080              ; 0081        0000          org    01ffh   ; set reset vector
0082    01FF  0A5E           goto    PWRUP
0083          0000           org     000h    ;
0084    0000  0A5E           goto    PWRUP
0085                ;
0086             ;************************************************************* 0087          ;
DELAY ROUTINE
0088                ;       This routine takes the value in 'loops'
0089                ;       and multiplies it times 1 millisecond to
0090                ;       determine delay time.
0091                ;*************************************************************
0092               WAIT 0093              ; 0094   0001  0C6E  top      movlw   .110          ; timing
adjustment variable
0095    0002  0036            movwf   loops2
0096    0003  0000  top2      nop                     ; sit and wait
0097    0004  0000            nop
0098    0005  0000            nop
0099    0006  0000            nop
0100    0007  0000            nop
0101    0008  0000            nop
0102    0009  02F6            decfsz  loops2          ; inner loops complete?
 16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 3
Line    PC    Opcode

0103    000A  0A03            goto    top2            ; no, go again
0104                                                  ;
0105    000B  02F5            decfsz  loops           ; outer loops complete?
0106    000C  0A01            goto    top             ; no, go again
0107    000D  0800            retlw   0               ; yes, return from sub
0108                ;
0109             ;************************************************************* 0110          ;
Start Bit Subroutine
0111                ;       this routine generates a start bit
0112                ;       (Low going data line while clock is high)
0113                ;*************************************************************
0114                ;
0115               BSTART 0116   000E  05E6          bsf     port_b,sdata    ; make sure data is high
0117    000F  0C3F            movlw   b'00111111'
0118    0010  0006            tris    port_b          ; set data and clock lines for output
0119    0011  04C6            bcf     port_b,sclk     ; make sure clock is low
0120    0012  0000            nop
0121    0013  05C6            bsf     port_b,sclk     ; set clock high
0122    0014  0000            nop
0123    0015  0000            nop
0124    0016  0000            nop
```


© 1994 Microchip Technology Inc.

```
0125   0017  0000           nop
0126   0018  0000           nop
0127   0019  04E6           bcf     port_b,sdata    ; data line goes low during
0128                                                 ; high clock for start bit
0129   001A  0000           nop
0130   001B  0000           nop
0131   001C  0000           nop
0132   001D  0000           nop
0133   001E  0000           nop                     ; timing adjustment
0134   001F  04C6           bcf     port_b,sclk     ; start clock train
0135   0020  0000           nop
0136   0021  0000           nop
0137   0022  0800           retlw   0
0138               ;
0139               ;*************************************************************
0140               ;       Stop Bit Subroutine
0141               ;       This routine generates a stop bit
0142               ;       (High going data line while clock is high)
0143               ;*************************************************************
0144               BSTOP 0145   0023  0C3F          movlw   b'00111111'     ;
0146   0024  0006           tris    port_b          ; set data/clock lines as outputs
0147   0025  04E6           bcf     port_b,sdata    ; make sure data line is low
0148   0026  0000           nop
0149   0027  0000           nop
0150   0028  0000           nop
0151   0029  05C6           bsf     port_b,sclk     ; set clock high
0152   002A  0000           nop
0153   002B  0000           nop
 16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 4
Line   PC    Opcode

0154   002C  0000           nop
0155   002D  05E6           bsf     port_b,sdata    ; data goes high while clock high
0156                                                 ; for stop bit
0157   002E  0000           nop
0158   002F  0000           nop
0159   0030  04C6           bcf     port_b,sclk     ; set clock low again
0160   0031  0000           nop
0161   0032  0000           nop
0162   0033  0000           nop
0163   0034  0800           retlw   0
0164               ;
0165             ;************************************************************ 0166             ;
BITOUT routine takes one bit of data in 'do' and
0167               ;       transmits it to the serial EE device
0168               ;*************************************************************
0169             BITOUT 0170   0035  0C3F          movlw   b'00111111'     ; set data,clock as outputs
0171   0036  0006           tris    port_b
0172   0037  07CA           btfss   eeprom,do       ; check for state of data bit to xmit
0173   0038  0A3B           goto    bitlow          ;
0174   0039  05E6           bsf     port_b,sdata    ; set data line high
0175   003A  0A3C           goto    clkout          ; go toggle the clock
0176
0177   003B  04E6  bitlow   bcf     port_b,sdata    ; output a low bit
0178   003C  05C6  clkout   bsf     port_b,sclk     ; set clock line high
0179   003D  0000           nop
0180   003E  0000           nop
0181   003F  0000           nop
0182   0040  0000           nop
0183   0041  04C6           bcf     port_b,sclk     ; return clock line low
0184   0042  0800           retlw   0
0185               ;
0186             ;************************************************************ 0187             ;
BITIN routine reads one bit of data from the 0188               ;       serial EE device and stores
it in 'di'
0189               ;*************************************************************
0190             BITIN 0191   0043  05EA          bsf     eeprom,di       ; assume input bit is high
0192   0044  0CBF           movlw   b'10111111'     ; make sdata an input line
0193   0045  0006           tris    port_b
```


© 1994 Microchip Technology Inc.

```
0194   0046  05E6          bsf     port_b,sdata   ; set sdata line for input
0195   0047  05C6          bsf     port_b,sclk    ; set clock line high
0196   0048  0000          nop                    ; just sit here a sec
0197   0049  0000          nop
0198   004A  0000          nop
0199   004B  0000          nop
0200   004C  0000          nop                    ;
0201   004D  07E6          btfss   port_b,sdata   ; read the data bit
0202   004E  04EA          bcf     eeprom,di      ; input bit was low
0203   004F  04C6          bcf     port_b,sclk    ; set clock line low
0204                                              ;
 16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 5
Line   PC    Opcode

0205   0050  0800          retlw   0              ;
0206               ;
0207               ;*****************************************************************
0208               ;       Transmit Data Subroutine
0209               ;       This routine takes the byte of data stored in the
0210               ;       'datao' register and transmits it to the serial EE device.
0211               ;       It will then send 1 more clock to the serial EE for the
0212               ;       acknowledge bit.  If the ack bit from the part was low
0213               ;       then the transmission was sucessful.  If it is high, then
0214               ;       the device did not send a proper ack bit and the ack
0215               ;       fail LED will be turned on.
0216               ;*****************************************************************
0217               TX 0218   0051  0C08          movlw   .8
0219   0052  0031          movwf   count          ; set the #bits to 8
0220                        ;
0221               TXLP 0222   0053  04CA          bcf     eeprom,do      ; assume bit out is low
0223   0054  06F0          btfsc   txbuf,7        ; is bit out really low?
0224   0055  05CA          bsf     eeprom,do      ; otherwise data bit =1
0225   0056  0935          call    BITOUT         ; serial data out
0226   0057  0370          rlf     txbuf          ; rotate txbuf left
0227   0058  02F1          decfsz  count          ; 8 bits done?
0228   0059  0A53          goto    TXLP           ; no - go again
0229   005A  0943          call    BITIN          ; read ack bit
0230   005B  06EA          btfsc   eeprom,di      ; check ack bit
0231   005C  0525          bsf     port_a,ackf    ; set acknowledge fail LED if the
0232                        ;
0233   005D  0800          retlw   0
0234               ;
0235               ;*****************************************************************
0236               ;       Power up routine
0237               ;       This is the program entry point, which in this case simply
0238               ;       sets the port_a I/O lines and directs control to the
0239               ;       byte write routine.
0240               ;*****************************************************************
0241               PWRUP 0242   005E  0C00           movlw  b'00000000'
0243   005F  0005            tris   port_a          ; set port A as all output
0244   0060  0065            clrf   port_a          ; all output lines low
0245   0061  0A62            goto   set_sec         ; set go set the high endurance block
0246
0247               ;*****************************************************************
0248               ;       Set Security Routine
0249               ;       This routine sets the secure portion of the array to the
0250               ;       lower 4 blocks (2K bytes) in the array and then delays
0251               ;       about a half second.  This routine will repeat forever.
0252               ;*****************************************************************
0253              ; 0254              set_sec 0255             ;                        16c5x/
7x Cross-Assembler V4.12 Released  Mon Jun 06 10:50:03 1994  Page 6
Line   PC    Opcode

0256   0062  0065          clrf    port_a         ; clear all LEDs
0257   0063  0C00          movlw   .0
0258   0064  0038          movwf   strt_blk       ; set the first protected block as
0259                                              ; block 0
0260   0065  0C04          movlw   .4
0261   0066  0039          movwf   sec_blks       ; set the number of protected blocks
```

8

```
0262                                    ; as 4 (2048 bytes)
0263   0067  0378        rlf     strt_blk        ; rotate starting block left 1 bit
0264                                    ; to arrange it correctly
0265   0068  05F8        bsf     strt_blk,7      ; set msb bit in starting block address
0266   0069  05F9        bsf     sec_blks,7      ; set msb bit block count byte
0267   006A  090E        call    BSTART          ; generate start bit
0268   006B  0CA0        movlw   b'10100000'     ; get slave address and write mode
0269   006C  0030        movwf   txbuf           ; into transmit buffer
0270   006D  0951        call    TX              ; and send it
0271
0272   006E  0218        movf    strt_blk,w      ; get the starting block address
0273   006F  0030        movwf   txbuf           ; into transmit buffer
0274   0070  0951        call    TX              ; and send it
0275   0071  0070        clrf    txbuf           ; clear buffer
0276   0072  0951        call    TX              ; send 8 don't care bits
0277
0278   0073  0219        movf    sec_blks,w      ; get secure blocks byte
0279   0074  0030        movwf   txbuf           ; into transmit buffer
0280   0075  0951        call    TX              ; and send it
0281   0076  0923        call    BSTOP           ; send stop bit
0282
0283              0284  0077  0CFF        movlw   .255            ; long delay
0285   0078  0035        movwf   loops
0286   0079  0901        call    wait
0287   007A  0CFF        movlw   .255
0288   007B  0035        movwf   loops
0289   007C  0901        call    wait
0290
0291   007D  0A62        goto    set_sec         ; go again
0292              ;
0293            ; 0294      0000  END                       16c5x/7x Cross-Assembler V4.12 Released  Mon
Jun 06 10:50:03 1994  Page 7
```


© 1994 Microchip Technology Inc.

MICROCHIP

AN517

24C01A Compatibility Issue and Its Mobility for Memory Upgrade

INTRODUCTION

The 24C01A is a 1K (128 x 8) Serial EEPROM which is currently offered by Microchip and Xicor. There are several important differences between the two devices which are discussed in this report. This report refers to the Microchip part as the 24C01A and the Xicor part as the X24C01. It is intended to assist in designing a memory subsystem which is compatible with either device.

COMPATIBILITY ISSUES

There are three major differences between Microchip's 24C01A and Xicor's X24C01 as detailed below.

1.1 PAGE MODE DIFFERENCES

The 24C01A was originally designed to work in the same socket as the PCD8572 which has a two-byte page mode. Therefore, its page buffer is two bytes deep. The X24C01 has a page mode of four bytes depth.

If more than two bytes are transmitted to the 24C01A during a page programming cycle, the 24C01A will terminate the write cycle.

In many applications where serial EEPROMs are used and speed is not a key issue, the byte write mode can be used without any loss of system performance. If only the byte write mode is used, there is no compatibility problem (other than the slave address software differences discussed in 1.2).

If the page write feature must be used, two different page mode algorithms can be transmitted by the master depending upon whose device is being used. The master will have to first do a polling routine to determine if it is interfacing with a 24C01A or X24C01. This polling technique is discussed in 1.2.

Interestingly, the 24C01A actually updates faster in the page mode even though it has one-half the page depth of the X24C01. This is due to the faster write cycle time of the 24C01A. The two devices are compared in Figure 1.

1.2 SOFTWARE DIFFERENCES

Microchip's 24C01A is designed to share a 2-wire bus on which it resides with other devices. To support this, the first byte of each command sequence from the master to the 24C01A must be a slave address. The 24C01A monitors the 2-wire bus for its slave address and "wake-up" from standby mode if the address transmitted matches its address as defined by the voltage level (Vss or Vcc) on pins 1, 2 and 3. X24C01 does not support a multiple device bus and will always "wake-up" if a start condition is detected.

A slave address must be transmitted to the 24C01A at certain points during reading and writing. This slave address is not required by the X24C01. Transmitting a slave address to X24C01 will result in erroneous operation. This problem can be solved by having the master transmit the proper serial bit pattern to the slave, but first the master has to ascertain with which 24C01A it is communicating.

The master can do a simple polling routine before beginning serial communication with 24C01A or X24C01 to determine with which device it is working. The proper serial protocol for both devices must be contained in the master controller's firmware. Once the master knows which 24C01A is on the bus, it can execute the proper serial commands.

8

FIGURE 1 - PAGE MODE DIFFERENCES

	Microchip	*Xicor*
Max byte program time	1 ms	10 ms
Max page program time	2 ms (2 bytes)	10 ms (4 bytes)
Max time to program 4 bytes	4 ms	10 ms
Max time to rewrite device	128 ms	320 ms

© 1992 Microchip Technology Inc.

The polling consists of the pattern like the one shown below:

| SDA LINE: | START BIT | 00000001 | ACKNOWLEDGE BIT | DATA 7...0 |

If an X24C01 is used on the 2-wire bus, an acknowledge bit and eight data bits will be returned whereas 24C01A will issue no response and will ignore the command.

2.3 HARDWARE DIFFERENCES

Unlike the X24C01, the 24C01A is designed to share a 2-wire bus with other devices. Chip address bits are included in the slave address for the 24C01A, and are incorporated into pins 1, 2 and 3 of the device. They must be connected to Vcc or Vss for proper operation. Since pins 1, 2 and 3 of the Xicor part are NC (no connect) pins and they are not internally connected, they can be tied high or low.

Another hardware difference involves pin 7 which *MUST* be connected to Vss on the X24C01. The 24C01A can have pin 7 connected to Vss or Vcc.

If only one device is planned for the 2-wire bus, the board can be made compatible for either device by connecting pins 1, 2 and 3 to either Vss or Vcc and tying pin 7 to Vss.

Mobility For Memory Upgrade And Expansion

In system applications where the master device needs to address more than one serial EEPROM on a 2-wire bus, the Microchip 24C01A offers the flexibility. Up to eight 24C01A's can be connected to the 2-wire bus. More than one Xicor X24C01 connected to the bus may result in bus contention.

If memory upgrade is required, the Microchip 24C01A can be upgraded to the 24C02A (256 x 8) or the 24C04A (512 x 8) in the same IC socket with *NO change* in hardware. Using the Xicor X24C01, both software and hardware would have to be reconfigured to accommodate the changes.

Author: *Bruce Negley*
Memory Products Division

© 1992 Microchip Technology Inc.

MICROCHIP

AN559

Optimizing Serial Bus Operations with Proper Write Cycle Times

SERIAL EEPROM WRITE TIME REQUIREMENTS

Elements of the Write Cycle Time

The total write operation time for a Serial EEPROM is determined by three main elements:

- Number of bytes to load for each write operation
- Bus clock speed at which the write operation is loaded
- Fixed internal write cycle timer required for the programming operation

The load component of the write command consists of the control byte, address, and the data of up to 16 bytes. The time required to load the operation depends on the number of bytes to load at one time and the bus clock speed. After this load is complete, the part commences the internally controlled write cycle and the bus and system are free to perform other tasks. The internal write cycle timer is a fixed time delay which is required to program the EEPROM memory cells. Table 1 gives examples of total write time for 1 and 16 bytes for various parts at fast and normal clock speeds.

- **Load time (time at bus free)** is the time the part is being loaded with the instruction, address, and data. The bus is free after this time interval, and the part commences the internally controlled write cycle sequence.
- **Write timer (worse case)** indicates the time the part is in the internally controlled write cycle allowing for the maximum specified datasheet requirements.
- **Write timer (typical)** indicates the time the part is in the internally controlled write cycle assuming nominal conditions and utilizing write cycle polling.
- **Total write** time is the combined load time and typical internal write cycle time.

MINIMIZING SERIAL BUS COMMUNICATION TIME IN A SYSTEM

Utilizing the Page Write Option

The original Microchip Serial EEPROM products, though utilizing a page buffer, only write bytes sequentially. This means the time required to write 8 bytes is 8 ms, worse case. The new 24LCXX products incorporate a page mode that allows simultaneous writes of up to 16 bytes. This allows the programming of up to 16 bytes in one write cycle (10 ms) compared to 16 write cycles (16 ms) for the original 24CXX products. Microchip uses an 8 byte page in the 24LC01 and 24LC02, and a 16 byte page in the 24LC04 and higher densities. What this means to a system designer is a write of 8 bytes in a 24LC01 or 24LC02 would take 10 ms, worse case,

TABLE 1 - WRITE OPERATION TIME COMPARISON

Product	Page Width	# Bytes to load	Speed (KHz)	Load time (ms) time at bus free	Write timer (ms) Worse case	Write timer (ms) Typical (25C)	Typical total write time (ms)
24C01A	2	1	100	0.28	1	0.4	0.68
		8 (4 x 2)	100	1.12	1	0.4	4.32
24LC01	8	1	100	0.28	10	3	3.28
		8	100	0.91	10	3	3.91
		1	400	0.07	10	3	3.07
		8	400	0.23	10	3	3.23
24C04A	8	1	100	0.28	1	0.4	0.68
		16 (2 x 8)	100	1.82	1	0.4	8.22
24LC04	16	1	100	0.28	10	3	3.28
		16	100	1.63	10	3	4.63
		1	400	0.07	10	3	3.07
		16	400	0.41	10	3	3.41

8

© 1993 Microchip Technology Inc.

versus 8 ms, worse case, in either a 24C01A or 24C02A. Loading the entire memory of an original 24C04A takes 512 ms, but the same operation on a new 24LC04 is reduced to only 320 ms, assuming worse case. As Table 1 shows, the difference is even greater using typical numbers.

The 93LCXX products do not utilize a page mode. However, in 16 bit mode, all 16 bits are written simultaneously. The original 93CXX products write the 16 bits in 2 write cycles, so the write cycle time increase of the new products is only 5 X (3.5 X typical).

Utilizing Write Cycle Polling

One powerful method of increasing programming efficiency is by periodically polling the part to determine if the write cycle has completed. To poll the 24LCXX series products, a control byte is sent and the acknowledge bit from the part is read. If the part acknowledges (pulls SDA low), it is ready to accept a new command. The part will not acknowledge while in the internally timed write cycle.

To poll the 93LCXX series products, the chip select is pulled high after the write cycle commences, and the data out line is read for the ready/busy status. If the part is still busy, it will pull the data line low. When the internally timed write cycle is complete, the part will pull the data line high, indicating it is ready for a new command.

Serial EEPROM System Optimization

Serial EEPROMs are used in systems for two purposes: storing data and reading back data. Read operations are at full clock speed, so the only methods for optimization are to run the clock at maximum frequency and to utilize sequential read whenever possible. Sequential read allows a continuous output data stream on one command.

The write operation, with its internal write timer component, needs special consideration when designing the control software. Efficient operation can be accomplished using both the page mode and write cycle polling. The following example shows a typical fetch-store operation in a system and how optimization can be incorporated. In the example system, the microcontroller must fetch bytes of data from a sensor and send the bytes to a EEPROM for storage. Two cases are shown in Figure 2: case 1 uses a 24C01A with no optimization; case 2 uses a 24LC01B with the available page mode and write cycle polling.

By utilizing the available page write mode and by polling for the write cycle completion, nearly four times as many bytes can be initially loaded to the serial EEPROM in the same time interval. Continuous operation for the optimized case 2 takes only 0.49 ms per byte compared to 1.28 ms per byte for the non-optimized case 1.

© 1993 Microchip Technology Inc.

FIGURE 2 - CASE COMPARISON

CASE 1 (NO OPTIMIZATION):		**CASE 2** (USING PAGE MODE AND POLLING):	
Operation	Serial Bus Time	Operation	Serial Bus Time
fetch byte		fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		load 8 bytes to serial	0.91 ms
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		poll for write timer	<3 ms typical
wait for write timer	1 ms	load 8 bytes to serial	0.91 ms
load to serial	0.28 ms		
fetch byte			
wait for write timer	1 ms		
load to serial	0.28 ms		
First **8 bytes** loaded in 9.24 ms		First **16 bytes** loaded in 4.82 ms	

8

Author: *Lenny French*
Memory Products Division

© 1993 Microchip Technology Inc.

NOTES:

© 1993 Microchip Technology Inc.

Using the 93LC56 and 93LC66

INTRODUCTION

The Microchip Technology Inc. 93LC56/66 are low-power 3-wire non-volatile memories and are suitable for many embedded system code and data storage applications. These devices are easily interfaced to most microcontrollers in today's market place, but Microchip's 8-bit RISC series PIC16CXX offers the best code density of any microcontroller on the market today. Using the PIC16C54, the assembly programs contained in this application note have been fully tested and provide the correct timing and 3-wire sequences to fully operate the 93LC56/66 in a PIC16CXX-based embedded application. The PIC16C54 was clocked at a 10MHz frequency. This application note is intended to provide the engineer with readily available stand-alone code modules to accomplish all of the necessary functions to utilize these devices in a low power application using the efficient PIC16C54 microcontroller.

The 93 series of devices have essentially four I/O pins:

CS	Chip Select
CLK	Clock
DI	Data In
DO	Data Out

This series of devices use a series of commands to accomplish the normal memory functions. These are READ, WRITE, EWEN, ERASE, ERAL, WRAL, EWDS. For a more detailed discussion of the function of these devices reference the appropriate data sheet and AN536, also published by Microchip Technology.

The following programs are included in this application note and are fully functional stand-alone modules. They are intended for use by those who are not already familiar with interfacing a PIC16CXX microcontroller to a 93 series device. For those with more experience, please refer to application note AN530.

3-Wire Byte Read Program

- Start Bit Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

3-Wire Byte Write Program

- Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Byte Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Byte Write with Data Polling Program

- Data Polling Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine
- Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Sequential Read Program

- Delay Routine
- Start Bit Routine
- Bit In Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

Author: *Bruce Negley*
Memory Products Division

8

© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:10 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*****************************************************************
0003                ;       3-Wire Byte Read Program (80 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
0007                ;       device.  This program will read 8 consecutive addresses
0008                ;       in the 'random read' mode.  This means that the opcode
0009                ;       and address for each byte will be sent to the device.
0010                ;       This program will repeat forever.
0011                ;
0012                ;       Another, more efficient method of reading consecutive
0013                ;       addresses is called the 'sequential read' mode.  This
0014                ;       involves sending the opcode and address for the first
0015                ;       byte to read, then continuing to provide clocks for the
0016                ;       next addresses. The device will automatically increment
0017                ;       the address.  An example of the sequential read mode is
0018                ;       provided in the '3wseqr.asm' file.
0019                ;
0020                ;       This program communicates to the serial EE in the
0021                ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
0022                ;       ON THE DEVICE TO Vcc.
0023                ;
0024                ;       Timing is based on using the PIC16C54 in 'XT' mode
0025                ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0026                ;       will be approximately 50 kHz for this setup.
0027                ;
0028                ;        PIC16C54 to Serial EE Connections:
0029                ;
0030                ;         PIC16C54          Serial EE
0031                ;        ----------        ----------
0032                ;       Pin 10 (RB4) -> Chip Select
0033                ;       Pin 11 (RB5) -> Clock
0034                ;       Pin 12 (RB6) -> Data In
0035                ;       Pin 13 (RB7) -> Data Out
0036                ;                       ORG = Vcc
0037                ;
0038                ;*****************************************************************
0039                ;               Register Assignments
0040                ;*****************************************************************
0041          0003  status  equ     3h      ; status register
0042          0005  port_a  equ     5h      ; port 5 (port_a)
0043          0006  port_b  equ     6h      ; port 6 (port b) comm lines to serial EE
0044          000A  eeprom  equ     0ah     ; bit buffer
0045          000C  addr    equ     0ch     ; address register
0046          000D  datai   equ     0dh     ; stored data input reg.
0047          000E  datao   equ     0eh     ; stored data output reg.
0048          0010  txbuf   equ     10h     ; transmit buffer
0049          0011  count   equ     11h     ; bits transmitted so far
0050          0012  bits    equ     12h     ; bits to transmit
0051          0013  bytcnt  equ     13h     ; byte counter for read routine

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:10 1994  Page 2

Line   PC    Opcode

0052          0015  loops   equ     15h     ; delay loop counter
0053          0016  loops2  equ     16h     ; delay loop counter
0054          0017  hbyte   equ     17h     ; high byte for input data
0055          0018  lbyte   equ     18h     ; low byte for input data
0056                ;*****************************************************************
0057                ;               Bit Assignments
0058                ;*****************************************************************
0059          0007  di      equ     7       ; eeprom input
0060          0006  do      equ     6       ; eeprom output
```


© 1994 Microchip Technology Inc.

```
0061          0007  datout  equ    7        ; data out line (port_b)
0062          0006  datin   equ    6        ; data in line (port_b)
0063          0005  sclk    equ    5        ; clock line (port_b)
0064          0004  chpsel  equ    4        ; chip select line (port_b)
0065                ;
0066                ;*****************************************************************
0067          0000          org    01ffh
0068    01FF  0A33  begin   goto   PWRUP    ; set the reset vector
0069          0000          org    000h
0070    0000  0A33          goto   PWRUP
0071                ;
0072                ;*****************************************************************
0073                ;       Start Bit Subroutine
0074                ;       this routine generates a start bit
0075                ;       (Chip select and DI high when clock goes high)
0076                ;*****************************************************************
0077                BSTART
0078    0001  04C6            bcf     port_b,datin    ; set datain and chipselect lines
0079    0002  0486            bcf     port_b,chpsel   ; low just to check operation
0080    0003  04A6            bcf     port_b,sclk     ; make sure clock starts low too.
0081    0004  0000            nop
0082                ;
0083    0005  0586            bsf     port_b,chpsel   ; set chip select line high
0084    0006  05C6            bsf     port_b,datin    ; set data in line high
0085    0007  0000            nop
0086    0008  05A6            bsf     port_b,sclk     ; set the clock line high to
0087                                                  ; generate the start bit
0088    0009  0000            nop
0089    000A  0000            nop
0090    000B  04A6            bcf     port_b,sclk     ; set clock low again
0091    000C  0800            retlw   0
0092                          ;
0093                ;*****************************************************************
0094                ;       BITIN routine reads one bit of data from the
0095                ;       serial EE device and stores it in 'di'
0096                ;*****************************************************************
0097                BITIN
0098    000D  05EA            bsf     eeprom,di       ; assume input bit is high
0099    000E  05A6            bsf     port_b,sclk     ; set clock line high
0100    000F  0000            nop                     ;
0101    0010  07E6            btfss   port_b,datout   ; read the data bit
0102    0011  04EA            bcf     eeprom,di       ; input bit was low

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:10 1994  Page 3

Line    PC    Opcode

0103    0012  04A6            bcf     port_b,sclk     ; set clock line low
0104                                                  ;
0105    0013  0800            retlw   0               ;
0106
0107                ;*****************************************************************
0108                ;       Receive data routine
0109                ;       This routine reads one byte of data from the part
0110                ;       into the 'datai' register.
0111                ;*****************************************************************
0112                RX
0113    0014  006D            clrf    datai           ; clear input buffer
0114    0015  0C08            movlw   .8              ; set # bits to 8
0115    0016  0031            movwf   count
0116    0017  0403            bcf     status,0        ; make sure carry bit is low
0117    0018  036D  RXLP      rlf     datai           ; rotate the buffer left 1 bit
0118    0019  090D            call    BITIN           ; read 1 bit
0119    001A  040D            bcf     datai,0         ; assume the input bit was low
0120    001B  06EA            btfsc   eeprom,di       ; check the bit
0121    001C  050D            bsf     datai,0         ; set high if neccessary
0122    001D  02F1            decfsz  count           ; 8 bits done?
0123    001E  0A18            goto    RXLP            ; no, do another
0124    001F  0800            retlw   0
```

8

© 1994 Microchip Technology Inc.

```
0125                    ;
0126                    ;*****************************************************************
0127                    ;       BITOUT routine
0128                    ;       This routine takes one bit of data in 'do' and
0129                    ;       transmits it to the serial EE device
0130                    ;*****************************************************************
0131                    BITOUT
0132    0020  07CA              btfss   eeprom,do       ; check state of data bit
0133    0021  0A24              goto    bitlow          ; low, goto bitlow
0134    0022  05C6              bsf     port_b,datin    ; high, set datain high
0135    0023  0A25              goto    clkout          ; and clock it
0136                            ;
0137    0024  04C6  bitlow      bcf     port_b,datin    ; output a logic low
0138    0025  05A6  clkout      bsf     port_b,sclk     ; set clock line high
0139    0026  0000              nop
0140    0027  04A6              bcf     port_b,sclk     ; return clock line low
0141    0028  0800              retlw   0
0142                    ;*****************************************************************
0143                    ;       Transmit Data Subroutine
0144                    ;       This routine takes the byte of data stored in the
0145                    ;       'datao' register and transmits it to the serial EE device.
0146                    ;*****************************************************************
0147                    TX
0148    0029  0212              movf    bits,w          ; set the number of bits to xmit
0149    002A  0031              movwf   count
0150                            ;
0151                    TXLP
0152    002B  04CA              bcf     eeprom,do       ; assume bit 7 is low

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:10 1994  Page 4

Line    PC    Opcode

0153    002C  06F0              btfsc   txbuf,7         ; is bit 7 clear?
0154    002D  05CA              bsf     eeprom,do       ; no, set data bit =1
0155    002E  0920              call    BITOUT          ; transmit 1 bit to serial EE
0156    002F  0370              rlf     txbuf           ; rotate txbuf left
0157    0030  02F1              decfsz  count           ; all bits done?
0158    0031  0A2B              goto    TXLP            ; no, do another bit
0159    0032  0800              retlw   0               ; yes, jump out
0160                    ;
0161                    ;*****************************************************************
0162                    ;       POWER-UP ROUTINE
0163                    ;       This is the program entry point, which in this case simply
0164                    ;       sets the port_a I/O lines and directs control to the
0165                    ;       read routine.
0166                    ;*****************************************************************
0167                    PWRUP
0168                    ;
0169    0033  0C00              movlw   b'00000000'
0170    0034  0005              tris    port_a          ; set port_a as all output
0171    0035  0065              clrf    port_a          ; all lines low
0172                            ;
0173    0036  0C80              movlw   b'10000000'
0174    0037  0006              tris    port_b          ; set RB7 as input, rest output;
0175                    ;
0176                    ;       Fall through and do the read
0177                    ;
0178                    ;*****************************************************************
0179                    ;       READ ROUTINE
0180                    ;       This routine reads 8 consecutive addresses in
0181                    ;       random mode starting at address 0.  This is done in
0182                    ;       x16 mode and will repeat forever.
0183                    ;*****************************************************************
0184                    READ
0185                    ;
0186    0038  0C00              movlw   .0              ; set starting address to 00
0187    0039  002C              movwf   addr            ;
0188
```


© 1994 Microchip Technology Inc.

```
0189   003A  0C08          movlw   .8             ; set number of addresses to
0190   003B  0033          movwf   bytcnt         ; read as 8
0191                                              ;
0192   003C  0901  rbyte    call    BSTART         ; generate the start bit
0193   003D  0C02          movlw   .2             ; set # bits to 2
0194   003E  0032          movwf   bits           ;
0195   003F  0C80          movlw   b'10000000'    ; get opcode (10b)
0196   0040  0030          movwf   txbuf          ; into output buffer
0197   0041  0929          call    TX             ; and transmit it
0198   0042  0C08          movlw   .8             ;
0199   0043  0032          movwf   bits           ; set number of bits to 8
0200   0044  020C          movf    addr,w         ; get the address
0201   0045  0030          movwf   txbuf          ; into the output buffer
0202   0046  0929          call    TX             ; and transmit it
0203
0204   0047  0914          call    RX             ; read the high byte

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:10 1994  Page 5

Line   PC    Opcode

0205   0048  020D          movf    datai,w        ; move input data to w
0206   0049  0037          movwf   hbyte          ; xfer it to high byte
0207
0208   004A  0914          call    RX             ; read the low byte
0209   004B  020D          movf    datai,w        ; move input data to w
0210   004C  0037          movwf   hbyte          ; xfer it to low byte
0211
0212   004D  0486          bcf     port_b,chpsel  ; clear the chip select line
0213
0214   004E  02AC          incf    addr           ; add 1 to the address
0215   004F  02F3          decfsz  bytcnt         ; have all bytes been read?
0216   0050  0A3C          goto    rbyte          ; no, read another byte
0217   0051  0A38          goto    READ           ; yes, start over
0218         0000  END
```

8

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002              ;*****************************************************************
0003              ;       3-Wire Byte Write Program (106 bytes)
0004              ;
0005              ;       This program demonstrates how to interface a
0006              ;       Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
0007              ;       device. This program will execute the erase/write enable
0008              ;       command, write to 8 consecutive addresses, and then
0009              ;       execute the erase/write disable command.  This
0010              ;       sequence will repeat forever.
0011              ;
0012              ;       After each byte is written, time must be given to the
0013              ;       device for it to complete the write cycle before
0014              ;       the next command can be sent.  The easiest solution
0015              ;       is to consult the data book for the maximum write
0016              ;       cycle time and just wait that long before the next
0017              ;       command is sent.  This program demonstrates that
0018              ;       solution.
0019              ;
0020              ;       Another, more efficient method of determining when the
0021              ;       write cycle is complete is called 'data polling.'  This
0022              ;       method is demonstrated in the program "3wdpoll."
0023              ;
0024              ;       This program communicates to the serial EE in the
0025              ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
0026              ;       ON THE DEVICE TO Vcc.
0027              ;
0028              ;       Timing is based on using the PIC16C54 in 'XT' mode
0029              ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0030              ;       will be approximately 40 kHz for this setup.
0031              ;
0032              ;        PIC16C54 to Serial EE Connections:
0033              ;
0034              ;        PIC16C54         Serial EE
0035              ;        --------         ---------
0036              ;       Pin 10 (RB4) -> Chip Select
0037              ;       Pin 11 (RB5) -> Clock
0038              ;       Pin 12 (RB6) -> Data In
0039              ;       Pin 13 (RB7) -> Data Out
0040              ;                       ORG=Vcc
0041              ;
0042              ;*****************************************************************
0043              ;                Register Assignments
0044              ;*****************************************************************
0045        0005  port_a  equ     5h      ; port 5 (port_a)
0046        0006  port_b  equ     6h      ; port 6 (port b) comm lines to serial EE
0047        000A  eeprom  equ    0ah      ; bit buffer
0048        000C  addr    equ    0ch      ; address register
0049        000D  datai   equ    0dh      ; stored data input reg.
0050        000E  datao   equ    0eh      ; stored data output reg.
0051        0010  txbuf   equ    10h      ; transmit buffer

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 2

Line   PC    Opcode

0052        0011  count   equ    11h      ; bits transmitted so far
0053        0012  bits    equ    12h      ; bits to transmit
0054        0013  bytcnt  equ    13h      ; byte counter for write routine
0055        0015  loops   equ    15h      ; delay loop counter
0056        0016  loops2  equ    16h      ; delay loop counter        ;
0057              ;*****************************************************************
0058              ;                Bit Assignments
0059              ;*****************************************************************
0060        0007  di      equ    7        ; eeprom input
```


© 1994 Microchip Technology Inc.

```
0061          0006  do      equ    6        ; eeprom output
0062          0007  datout  equ    7        ; data out line (port_b)
0063          0006  datin   equ    6        ; data in line (port_b)
0064          0005  sclk    equ    5        ; clock line (port_b)
0065          0004  chpsel  equ    4        ; chip select line (port_b)
0066                ;
0067                ;*****************************************************************
0068          0000          org    01ffh
0069    01FF  0A2F  begin   goto   PWRUP    ; set the reset vector
0070          0000          org    000h
0071    0000  0A2F          goto   PWRUP
0072                ;
0073                ;*****************************************************************
0074                ;       DELAY ROUTINE
0075                ;       This routine takes the value in 'loops'
0076                ;       and multiplies it times 1 millisecond to
0077                ;       determine delay time.
0078                ;*****************************************************************
0079                WAIT
0080                ;
0081    0001  0C6E  top       movlw   .110          ; timing adjustment variable
0082    0002  0036            movwf   loops2
0083    0003  0000  top2      nop                   ; sit and wait
0084    0004  0000            nop
0085    0005  0000            nop
0086    0006  0000            nop
0087    0007  0000            nop
0088    0008  0000            nop
0089    0009  02F6            decfsz  loops2        ; inner loops complete?
0090    000A  0A03            goto    top2          ; no, go again
0091                                                ;
0092    000B  02F5            decfsz  loops         ; outer loops complete?
0093    000C  0A01            goto    top           ; no, go again
0094    000D  0800            retlw   0             ; yes, return from sub
0095                ;
0096                ;*****************************************************************
0097                ;       Start Bit Subroutine
0098                ;       this routine generates a start bit
0099                ;       (Chip select and DI high when clock goes high)
0100                ;*****************************************************************;
0101                BSTART
0102    000E  0C8F            movlw   b'10001111'

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 3

Line    PC    Opcode

0103    000F  0006            tris    port_b        ; set port b for output
0104                                                ; except for the data out line
0105    0010  04C6            bcf     port_b,datin  ; set datain and chipselect lines
0106    0011  0486            bcf     port_b,chpsel ; low just to check operation
0107    0012  04A6            bcf     port_b,sclk   ; make sure clock starts low too.
0108    0013  0000            nop
0109                ;
0110    0014  0586            bsf     port_b,chpsel ; set chip select line high
0111    0015  05C6            bsf     port_b,datin  ; set data in line high
0112    0016  0000            nop
0113    0017  05A6            bsf     port_b,sclk   ; set the clock line high to
0114                                                ; generate the start bit
0115    0018  0000            nop
0116    0019  0000            nop
0117    001A  04A6            bcf     port_b,sclk   ; set clock low again
0118    001B  0800            retlw   0
0119                          ;
0120                ;*****************************************************************
0121                ;       BITOUT routine
0122                ;       This routine takes one bit of data in 'do' and
0123                ;       transmits it to the serial EE device
0124                ;*****************************************************************
```

8

```
0125                    BITOUT
0126   001C  07CA              btfss   eeprom,do       ; check state of data bit
0127   001D  0A20              goto    bitlow          ; low, goto bitlow
0128   001E  05C6              bsf     port_b,datin    ; high, set datain high
0129   001F  0A21              goto    clkout          ; and clock it
0130                                                    ;
0131   0020  04C6  bitlow      bcf     port_b,datin    ; output a logic low
0132   0021  05A6  clkout      bsf     port_b,sclk     ; set clock line high
0133   0022  0000              nop
0134   0023  04A6              bcf     port_b,sclk     ; return clock line low
0135   0024  0800              retlw   0
0136                    ;
0137                    ;*****************************************************************
0138                    ;       Transmit Data Subroutine
0139                    ;       This routine takes the byte of data stored in the
0140                    ;       'datao' register and transmits it to the serial EE device.
0141                    ;*****************************************************************
0142                    TX
0143   0025  0212              movf    bits,w          ; set the number of bits to xmit
0144   0026  0031              movwf   count
0145                            ;
0146                    TXLP
0147   0027  04CA              bcf     eeprom,do       ; assume bit 7 is low
0148   0028  06F0              btfsc   txbuf,7         ; is bit 7 clear?
0149   0029  05CA              bsf     eeprom,do       ; no, set data bit =1
0150   002A  091C              call    BITOUT          ; transmit 1 bit to serial EE
0151   002B  0370              rlf     txbuf           ; rotate txbuf left
0152   002C  02F1              decfsz  count           ; all bits done?
0153   002D  0A27              goto    TXLP            ; no, do another bit

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 4

Line   PC    Opcode

0154   002E  0800              retlw   0               ; yes, jump out
0155                    ;
0156                    ;*****************************************************************
0157                    ;           POWER-UP ROUTINE
0158                    ;       This is the program entry point, which in this case simply
0159                    ;       sets the port_a I/O lines and directs control to the
0160                    ;       erase/write enable routine.
0161                    ;*****************************************************************
0162                    PWRUP
0163                    ;
0164   002F  0C00              movlw   b'00000000'
0165   0030  0005              tris    port_a          ; set port_a as all output
0166   0031  0065              clrf    port_a          ; all lines low
0167
0168   0032  0C80              movlw   b'10000000'
0169   0033  0006              tris    port_b          ; set RB7 as input, rest output;
0170                    ;
0171                    ;       Fall through and do erase/write enable
0172                    ;
0173                    ;*****************************************************************
0174                    ;         EWEN (Erase/Write ENable Routine)
0175                    ;           this routine enables the dut for erasing and
0176                    ;           writing. This must be done prior to any erase,write
0177                    ;           eral,wral instructions.
0178                    ;*****************************************************************
0179                    EWEN
0180                    ;
0181   0034  090E              call    BSTART          ; generate a start bit
0182                                                    ;
0183   0035  0C02              movlw   .2              ; set # bits to 2
0184   0036  0032              movwf   bits            ;
0185   0037  0C00              movlw   b'00000000'     ; get the opcode (00b)
0186   0038  0030              movwf   txbuf           ; into the output buffer
0187   0039  0925              call    TX              ; and transmit it
0188   003A  0C08              movlw   .8              ; set # bits to 8
```


© 1994 Microchip Technology Inc.

```
0189   003B  0032          movwf   bits           ;
0190   003C  0CC0          movlw   b'11000000'    ; get opcode and address
0191                                              ;       (11XXXXXX)
0192   003D  0030          movwf   txbuf          ; into output buffer
0193   003E  0925          call    TX             ; and transmit it
0194   003F  0486          bcf     port_b,chpsel  ; set chip select line low
0195   0040  0000          nop
0196               ;
0197               ;       Now continue on to the write command
0198               ;
0199               ;*****************************************************************
0200               ;       Byte Write Routine
0201               ;       This routine writes an AA55h pattern into
0202               ;       8 consecutive addresses starting at address 00.
0203               ;       A delay of about 10ms is given after each byte
0204               ;       for the write cycle to complete.

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 5

Line   PC    Opcode

0205               ;       The write is done in the x16 mode: the user must
0206               ;       have the ORG pin tied to Vcc on the device
0207               ;*****************************************************************
0208               WRITE
0209               ;
0210   0041  0C00          movlw   .0             ; set starting address to 00
0211   0042  002C          movwf   addr           ;
0212   0043  0C08          movlw   .8             ; set number of bytes to write as 8
0213   0044  0033          movwf   bytcnt         ;
0214                                              ;
0215   0045  090E  topwr    call    BSTART         ; generate the start bit
0216                                              ;
0217   0046  0C02          movlw   .2             ; set # bits to 2 for the opcode
0218   0047  0032          movwf   bits           ;
0219   0048  0C40          movlw   b'01000000'    ; get opcode (01b)
0220   0049  0030          movwf   txbuf          ; into the transmit buffer
0221   004A  0925          call    TX             ; and send it
0222                                              ;
0223   004B  0C08          movlw   .8             ; set # of bits to 8 for the
0224   004C  0032          movwf   bits           ; address
0225   004D  020C          movf    addr,w         ; get address counter
0226   004E  0030          movwf   txbuf          ; into output buffer
0227   004F  0925          call    TX             ; and send it
0228   0050  0CAA          movlw   b'10101010'    ; get upper byte of data (AAh)
0229   0051  0030          movwf   txbuf          ; into the transmit buffer
0230   0052  0925          call    TX             ; and send it
0231   0053  0C55          movlw   b'01010101'    ; get lower byte of data (55h)
0232   0054  0030          movwf   txbuf          ; into transmit buffer
0233   0055  0925          call    TX             ; and send it
0234                                              ;
0235   0056  0486          bcf     port_b,chpsel  ; clear the chip select line
0236                                              ; to initiate write cycle
0237                                              ;
0238   0057  0C0A          movlw   .10            ;
0239   0058  0035          movwf   loops          ; set delay time to 10mS
0240   0059  0901          call    WAIT           ; and wait
0241                                              ;
0242   005A  02AC          incf    addr           ; increment address counter
0243   005B  02F3          decfsz  bytcnt         ; all bytes written?
0244   005C  0A45          goto    topwr          ; no, do another
0245                                              ; yes, go on
0246               ;
0247               ;     Now continue on to the erase/write disable command
0248               ;
0249               ;*****************************************************************
0250               ;       EWDS (Erase/Write Disable Routine)
0251               ;       This routine executes the erase/write disable command
0252               ;       which prevents the contents of the array from being
```


© 1994 Microchip Technology Inc.

```
0253                    ;       written to.
0254                    ;
0255                    ;*****************************************************************

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:06 1994  Page 6

Line   PC    Opcode

0256                  EWDS
0257                  ;
0258   005D  090E             call    BSTART          ; generate a start bit
0259                  ;
0260   005E  0C02             movlw   .2              ; set # bits to 2
0261   005F  0032             movwf   bits            ;
0262   0060  0C00             movlw   b'00000000'     ; get the opcode (00b)
0263   0061  0030             movwf   txbuf           ; into the output buffer
0264   0062  0925             call    TX              ; and transmit it
0265   0063  0C08             movlw   .8              ; set # bits to 8
0266   0064  0032             movwf   bits            ;
0267   0065  0C00             movlw   b'00000000'     ; get opcode and address
0268                                                   ;     (00XXXXXX)
0269   0066  0030             movwf   txbuf           ; into output buffer
0270   0067  0925             call    TX              ; and transmit it
0271   0068  0486             bcf     port_b,chpsel   ; set chip select line low
0272   0069  0000             nop
0273   006A  0A34             goto    EWEN            ; start all over
0274                  ;
0275          0000    END
```


© 1994 Microchip Technology Inc.

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*****************************************************************
0003                ;       3-Wire Byte Write with Data Poll Program (107 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
0007                ;       device. This program will execute the erase/write enable
0008                ;       command, write to 8 consecutive addresses, and then
0009                ;       use the 'data polling' method to determine when the
0010                ;       write cycle is complete.  The program then executes
0011                ;       the erase/write disable command.  This sequence will
0012                ;       repeat forever.
0013                ;
0014                ;       When writing to a 3-wire serial EE device, the
0015                ;       internally timed write cycle will begin after
0016                ;       the opcode, address and data are sent to the
0017                ;       device.  There are two ways to make sure that the
0018                ;       cycle is complete before you send it another command.
0019                ;       The simplest method is to simply wait for the maximum
0020                ;       write cycle time, which can be obtained from the data book.
0021                ;       A more efficient method is "Data polling."  This is
0022                ;       done by toggling the chip select line low and then
0023                ;       back high after the opcode, address and data are sent.
0024                ;       The chip select line must be low for at least 250ns
0025                ;       before bringing it high again.  The device will pull
0026                ;       the data out line low at that point, and set it high
0027                ;       when the write cycle is complete.  The user can then check
0028                ;       or "poll" the dataout line until it goes high before
0029                ;       sending the next command.
0030                ;
0031                ;       As an option, the user can connect a LED to pin 18
0032                ;       on the PIC16C54 (use about a 1K resistor in series) as a
0033                ;       timeout indicator.  This LED will come on if the
0034                ;       data polling is unsuccessful and the device being
0035                ;       programmed does not respond.  This can be tested by
0036                ;       simply running the program with no part in the socket.
0037                ;
0038                ;       This program communicates to the serial EE in the
0039                ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
0040                ;       ON THE DEVICE TO Vcc.
0041                ;
0042                ;       Timing is based on using the PIC16C54 in 'XT' mode
0043                ;       using a 4Mhz crystal.  Clock speeds to the serial EE
0044                ;       will be approximately 50 kHz for this setup.
0045                ;
0046                ;        PIC16C54 to Serial EE Connections:
0047                ;
0048                ;         PIC16C54          Serial EE
0049                ;         --------          ---------
0050                ;       Pin 10 (RB4) -> Chip Select
0051                ;       Pin 11 (RB5) -> Clock

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 2

Line   PC    Opcode

0052                ;       Pin 12 (RB6) -> Data In
0053                ;       Pin 13 (RB7) -> Data Out
0054                ;                       ORG=Vcc
0055                ;
0056                ;*****************************************************************
0057                ;               Register Assignments
0058                ;*****************************************************************
0059        0005    port_a  equ     5h      ; port 5 (port_a)
0060        0006    port_b  equ     6h      ; port 6 (port b) used comm lines to serial EE
```

8

```
0061          000A  eeprom  equ    0ah      ; bit buffer
0062          000C  addr    equ    0ch      ; address register
0063          000D  datai   equ    0dh      ; stored data input reg.
0064          000E  datao   equ    0eh      ; stored data output reg.
0065          0010  txbuf   equ    10h      ; transmit buffer
0066          0011  count   equ    11h      ; bits transmitted so far
0067          0012  bits    equ    12h      ; bits to transmit
0068          0013  bytcnt  equ    13h      ; byte counter for write routine
0069          0015  loops   equ    15h      ; delay loop counter
0070          0016  loops2  equ    16h      ; delay loop counter
0071                ;*************************************************************
0072                ;                  Bit Assignments
0073                ;*************************************************************
0074          0007  di      equ    7        ; eeprom input
0075          0006  do      equ    6        ; eeprom output
0076          0007  datout  equ    7        ; data out line (port_b)
0077          0006  datin   equ    6        ; data in line (port_b)
0078          0005  sclk    equ    5        ; clock line (port_b)
0079          0004  chpsel  equ    4        ; chip select line (port_b)
0080          0001  timeout equ    1        ; write cycle timeout warning (port_a)
0081                ;
0082                ;*************************************************************
0083          0000        org    01ffh
0084    01FF  0A26  begin goto     PWRUP    ; set the reset vector
0085          0000        org    000h
0086    0000  0A26        goto     PWRUP
0087                ;
0088                ;*************************************************************
0089                ;       DATA POLL DELAY ROUTINE
0090                ;       This routine delays 100 us is
0091                ;       used for delay between polls
0092                ;*************************************************************
0093                dpdelay
0094                ;
0095    0001  0C19    movlw   .25              ; timing adjustment variable
0096    0002  0036    movwf   loops2
0097    0003  0000  top nop
0098    0004  02F6    decfsz  loops2           ; loops complete?
0099    0005  0A03    goto    top              ; no, go again
0100                                           ;
0101    0006  0800    retlw   0                ; yes, return from sub
0102                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 3

Line    PC    Opcode

0103                ;*************************************************************
0104                ;       Start Bit Subroutine
0105                ;       this routine generates a start bit
0106                ;       (Chip select and DI high when clock goes high)
0107                ;*************************************************************;
0108                BSTART
0109    0007  04C6    bcf     port_b,datin   ; set datain and chipselect lines
0110    0008  0486    bcf     port_b,chpsel  ; low just to check operation
0111    0009  04A6    bcf     port_b,sclk    ; make sure clock starts low too.
0112    000A  0000    nop
0113                ;
0114    000B  0586    bsf     port_b,chpsel  ; set chip select line high
0115    000C  05C6    bsf     port_b,datin   ; set data in line high
0116    000D  0000    nop
0117    000E  05A6    bsf     port_b,sclk    ; set the clock line high to
0118                                         ; generate the start bit
0119    000F  0000    nop
0120    0010  0000    nop
0121    0011  04A6    bcf     port_b,sclk    ; set clock low again
0122    0012  0800    retlw   0
0123                  ;
0124                ;*************************************************************
```


© 1994 Microchip Technology Inc.

```
0125                    ;       BITOUT routine
0126                    ;       This routine takes one bit of data in 'do' and
0127                    ;       transmits it to the serial EE device
0128                    ;*************************************************************
0129                    BITOUT
0130    0013  07CA              btfss   eeprom,do       ; check state of data bit
0131    0014  0A17              goto    bitlow          ; low, goto bitlow
0132    0015  05C6              bsf     port_b,datin    ; high, set datain high
0133    0016  0A18              goto    clkout          ; and clock it
0134                            ;
0135    0017  04C6  bitlow      bcf     port_b,datin    ; output a logic low
0136    0018  05A6  clkout      bsf     port_b,sclk     ; set clock line high
0137    0019  0000              nop
0138    001A  04A6              bcf     port_b,sclk     ; return clock line low
0139    001B  0800              retlw   0
0140                    ;
0141                    ;*****************************************************************
0142                    ;       Transmit Data Subroutine
0143                    ;       This routine takes the byte of data stored in the
0144                    ;       'datao' register and transmits it to the serial EE device.
0145                    ;*****************************************************************
0146                    TX
0147    001C  0212              movf    bits,w          ; set the number of bits to xmit
0148    001D  0031              movwf   count
0149                            ;
0150                    TXLP
0151    001E  04CA              bcf     eeprom,do       ; assume bit 7 is low
0152    001F  06F0              btfsc   txbuf,7         ; is bit 7 clear?
0153    0020  05CA              bsf     eeprom,do       ; no, set data bit =1

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 4

Line    PC    Opcode

0154    0021  0913              call    BITOUT          ; transmit 1 bit to serial EE
0155    0022  0370              rlf     txbuf           ; rotate txbuf left
0156    0023  02F1              decfsz  count           ; all bits done?
0157    0024  0A1E              goto    TXLP            ; no, do another bit
0158    0025  0800              retlw   0               ; yes, jump out
0159                    ;
0160                    ;*****************************************************************
0161                    ;               POWER-UP ROUTINE
0162                    ;       This is the program entry point, which in this case simply
0163                    ;       sets the port_a I/O lines and directs control to the
0164                    ;       erase/write enable routine.
0165                    ;*****************************************************************
0166                    PWRUP
0167                    ;
0168    0026  0C00              movlw   b'00000000'
0169    0027  0005              tris    port_a          ; set port_a as all output
0170    0028  0065              clrf    port_a          ; all lines low
0171
0172    0029  0C80              movlw   b'10000000'
0173    002A  0006              tris    port_b          ; set RB7 as input, rest output;
0174                    ;
0175                    ;       Fall through and do erase/write enable
0176                    ;
0177                    ;*****************************************************************
0178                    ;       EWEN (Erase/Write ENable Routine)
0179                    ;           this routine enables the dut for erasing and
0180                    ;           writing. This must be done prior to any erase,write
0181                    ;           eral,wral instructions.
0182                    ;*****************************************************************
0183                    EWEN
0184                    ;
0185    002B  0907              call    BSTART          ; generate a start bit
0186                            ;
0187    002C  0C02              movlw   .2              ; set # bits to 2
0188    002D  0032              movwf   bits            ;
```


© 1994 Microchip Technology Inc.

```
0189   002E  0C00           movlw   b'00000000'    ; get the opcode (00b)
0190   002F  0030           movwf   txbuf          ; into the output buffer
0191   0030  091C           call    TX             ; and transmit it
0192   0031  0C08           movlw   .8             ; set # bits to 8
0193   0032  0032           movwf   bits           ;
0194   0033  0CC0           movlw   b'11000000'    ; get opcode and address
0195                                                ;     (11XXXXXX)
0196   0034  0030           movwf   txbuf          ; into output buffer
0197   0035  091C           call    TX             ; and transmit it
0198   0036  0486           bcf     port_b,chpsel  ; set chip select line low
0199   0037  0000           nop
0200                ;
0201                ;       Now continue on to the write command
0202                ;
0203                ;*****************************************************************
0204                ;       WRITE

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 5

Line   PC    Opcode

0205                ;       This routine writes a AA55h pattern into
0206                ;       8 consecutive addresses starting at address 00.
0207                ;       It will then 'poll' the data out line to determine
0208                ;       when the write cycle is complete.  If the cycle has
0209                ;       not completed within 20 ms, then it will continue
0210                ;       anyway and turn the timeout fail LED on.
0211                ;       The write is done in the x16 mode: the user must
0212                ;       have the ORG pin tied to Vcc on the device. This
0213                ;       program will repeat forever.
0214                ;*****************************************************************
0215                WRITE
0216                ;
0217   0038  0C00           movlw   .0             ; set starting address to 00
0218   0039  002C           movwf   addr           ;
0219   003A  0C08           movlw   .8             ; set number of bytes to write as 8
0220   003B  0033           movwf   bytcnt         ;
0221                                                ;
0222   003C  0907  topwr    call    BSTART         ; generate the start bit
0223                                                ;
0224   003D  0C02           movlw   .2             ; set # bits to 2 for the opcode
0225   003E  0032           movwf   bits           ;
0226   003F  0C40           movlw   b'01000000'    ; get opcode (01b)
0227   0040  0030           movwf   txbuf          ; into the transmit buffer
0228   0041  091C           call    TX             ; and send it
0229                                                ;
0230   0042  0C08           movlw   .8             ; set # of bits to 8 for the
0231   0043  0032           movwf   bits           ; address
0232   0044  020C           movf    addr,w         ; get address counter
0233   0045  0030           movwf   txbuf          ; into output buffer
0234   0046  091C           call    TX             ; and send it
0235   0047  0CAA           movlw   b'10101010'    ; get first half of data (AAh)
0236   0048  0030           movwf   txbuf          ; into the transmit buffer
0237   0049  091C           call    TX             ; and send it
0238   004A  0C55           movlw   b'01010101'    ; get second half of data (55h)
0239   004B  0030           movwf   txbuf          ; into transmit buffer
0240   004C  091C           call    TX             ; and send it
0241                                                ;
0242   004D  0486           bcf     port_b,chpsel  ; clear the chip select line
0243                                                ; to initiate write cycle
0244   004E  0000           nop
0245   004F  0586           bsf     port_b,chpsel  ; set the chip sel line high
0246                                                ; to begin data polling
0247   0050  0CC8           movlw   .200           ;
0248   0051  0035           movwf   loops          ; poll 100 times before timeout(about 20ms)
0249   0052  0901  polltop  call    dpdelay        ; wait 100us
0250   0053  06E6           btfsc   port_b,datout  ; is the data out line high?
0251   0054  0A59           goto    okpoll         ; yes-cycle complete: do another
0252   0055  02F5           decfsz  loops          ; has it timed out?
```


© 1994 Microchip Technology Inc.

```
0253   0056  0A52          goto    polltop         ; no, check again
0254   0057  0525          bsf     port_a,timeout; yes, set timeout LED and go on
0255   0058  0A5A          goto    badpoll                 ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:00 1994  Page 6

Line   PC    Opcode

0256
0257   0059  0425  okpoll  bcf     port_a,timeout; clear the timeout LED
0258   005A  0486  badpoll bcf     port_b,chpsel ; chip sel line back low
0259   005B  02AC          incf    addr            ; increment address counter
0260   005C  02F3          decfsz  bytcnt          ; all bytes written?
0261   005D  0A3C          goto    topwr           ; no, do another
0262                                               ; yes, go on
0263                ;
0264                ;  Now continue on to the erase/write disable command
0265                ;
0266                ;*****************************************************************
0267                ;          EWDS (Erase/Write Disable Routine)
0268                ;       This routine executes the erase/write disable command
0269                ;       which prevents the contents of the array from being
0270                ;       written to.
0271                ;
0272                ;*****************************************************************
0273                EWDS
0274                ;
0275   005E  0907           call    BSTART          ; generate a start bit
0276                ;
0277   005F  0C02           movlw   .2              ; set # bits to 2
0278   0060  0032           movwf   bits            ;
0279   0061  0C00           movlw   b'00000000'     ; get the opcode (00b)
0280   0062  0030           movwf   txbuf           ; into the output buffer
0281   0063  091C           call    TX              ; and transmit it
0282   0064  0C08           movlw   .8              ; set # bits to 8
0283   0065  0032           movwf   bits            ;
0284   0066  0C00           movlw   b'00000000'     ; get opcode and address
0285                                                ;      (00XXXXXX)
0286   0067  0030           movwf   txbuf           ; into output buffer
0287   0068  091C           call    TX              ; and transmit it
0288   0069  0486           bcf     port_b,chpsel   ; set chip select line low
0289   006A  0000           nop
0290   006B  0A2B           goto    EWEN            ; start all over
0291                ;
0292                ;
0293
0294         0000  END
```

8

```
16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:20 1994  Page 1

Line   PC    Opcode

0001                    LIST P=16C54,c=132
0002                ;*****************************************************************
0003                ;       3-Wire Sequential Read Program (93 bytes)
0004                ;
0005                ;       This program demonstrates how to interface a
0006                ;       Microchip PIC16C54 to a 93LC56 or 93LC66 Serial EE
0007                ;       device.  This program will read 8 consecutive addresses
0008                ;       in the 'sequential read' mode.  This means that the opcode
0009                ;       and address are sent for the first address only.  After
0010                ;       the first address is read, the Chip Select line is left
0011                ;       high and clocks for the remaining 7 bytes are sent to the
0012                ;       device. The device will automatically increment the address
0013                ;       pointer in this mode, allowing the user to read as many
0014                ;       consecutive addresses as needed. This program will repeat
0015                ;       forever.
0016                ;
0017                ;       This program communicates to the serial EE in the
0018                ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
0019                ;       ON THE DEVICE TO Vcc.
0020                ;
0021                ;       Timing is based on using the PIC16C54 in 'XT' mode
0022                ;       using a 4MHz crystal.  Clock speeds to the serial EE
0023                ;       will be approximately 50 kHz for this setup.
0024                ;
0025                ;        PIC16C54 to Serial EE Connections:
0026                ;
0027                ;        PIC16C54          Serial EE
0028                ;       ----------        -----------
0029                ;       Pin 10 (RB4) -> Chip Select
0030                ;       Pin 11 (RB5) -> Clock
0031                ;       Pin 12 (RB6) -> Data In
0032                ;       Pin 13 (RB7) -> Data Out
0033                ;                       ORG = Vcc
0034                ;
0035                ;*****************************************************************
0036                ;               Register Assignments
0037                ;*****************************************************************
0038         0003   status  equ     3h      ; status register
0039         0005   port_a  equ     5h      ; port 5 (port_a)
0040         0006   port_b  equ     6h      ; port 6 (port b) used comm lines to serial EE
0041         000A   eeprom  equ     0ah     ; bit buffer
0042         000C   addr    equ     0ch     ; address register
0043         000D   datai   equ     0dh     ; stored data input reg.
0044         000E   datao   equ     0eh     ; stored data output reg.
0045         0010   txbuf   equ     10h     ; transmit buffer
0046         0011   count   equ     11h     ; bits transmitted so far
0047         0012   bits    equ     12h     ; bits to transmit
0048         0013   bytcnt  equ     13h     ; byte counter for read routine
0049         0015   loops   equ     15h     ; delay loop counter
0050         0016   loops2  equ     16h     ; delay loop counter
0051         0017   hbyte   equ     17h     ; high byte for input data

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:20 1994  Page 2

Line   PC    Opcode

0052         0018   lbyte   equ     18h     ; low byte for input data
0053                ;*****************************************************************
0054                ;               Bit Assignments
0055                ;*****************************************************************
0056         0007   di      equ     7       ; eeprom input
0057         0006   do      equ     6       ; eeprom output
0058         0007   datout  equ     7       ; data out line (port_b)
0059         0006   datin   equ     6       ; data in line (port_b)
0060         0005   sclk    equ     5       ; clock line (port_b)
```


© 1994 Microchip Technology Inc.

```
0061          0004  chpsel  equ    4       ; chip select line (port_b)
0062                ;
0063                ;*************************************************************
0064          0000          org    01ffh
0065    01FF  0A40  begin   goto   PWRUP   ; set the reset vector
0066          0000          org    000h
0067    0000  0A40          goto   PWRUP
0068                ;
0069                ;*************************************************************
0070                ;       DELAY ROUTINE
0071                ;       This routine takes the value in 'loops'
0072                ;       and multiplies it times 1 millisecond to
0073                ;       determine delay time.
0074                ;*************************************************************
0075                WAIT
0076                ;
0077    0001  0C6E  top       movlw  .110          ; timing adjustment variable
0078    0002  0036            movwf  loops2
0079    0003  0000  top2      nop                  ; sit and wait
0080    0004  0000            nop
0081    0005  0000            nop
0082    0006  0000            nop
0083    0007  0000            nop
0084    0008  0000            nop
0085    0009  02F6            decfsz  loops2       ; inner loops complete?
0086    000A  0A03            goto    top2         ; no, go again
0087                                               ;
0088    000B  02F5            decfsz  loops        ; outer loops complete?
0089    000C  0A01            goto    top          ; no, go again
0090    000D  0800            retlw   0            ; yes, return from sub
0091                ;
0092                ;*************************************************************
0093                ;       Start Bit Subroutine
0094                ;       this routine generates a start bit
0095                ;       (Chip select and DI high when clock goes high)
0096                ;*************************************************************
0097                BSTART
0098    000E  04C6          bcf     port_b,datin   ; set datain and chipselect lines
0099    000F  0486          bcf     port_b,chpsel  ; low just to check operation
0100    0010  04A6          bcf     port_b,sclk    ; make sure clock starts low too.
0101    0011  0000          nop
0102                ;

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:20 1994  Page 3

Line    PC    Opcode

0103    0012  0586          bsf     port_b,chpsel  ; set chip select line high
0104    0013  05C6          bsf     port_b,datin   ; set data in line high
0105    0014  0000          nop
0106    0015  05A6          bsf     port_b,sclk    ; set the clock line high to
0107                                               ; generate the start bit
0108    0016  0000          nop
0109    0017  0000          nop
0110    0018  04A6          bcf     port_b,sclk    ; set clock low again
0111    0019  0800          retlw   0
0112                ;
0113                ;*************************************************************
0114                ;       BITIN routine reads one bit of data from the
0115                ;       serial EE device and stores it in 'di'
0116                ;*************************************************************
0117                BITIN
0118    001A  05EA          bsf     eeprom,di      ; assume input bit is high
0119    001B  05A6          bsf     port_b,sclk    ; set clock line high
0120    001C  0000          nop                    ;
0121    001D  07E6          btfss   port_b,datout  ; read the data bit
0122    001E  04EA          bcf     eeprom,di      ; input bit was low
0123    001F  04A6          bcf     port_b,sclk    ; set clock line low
0124                                               ;
```

8

© 1994 Microchip Technology Inc.

```
0125   0020  0800          retlw   0               ;
0126
0127                 ;*****************************************************************
0128                 ;       Receive data routine
0129                 ;       This routine reads one byte of data from the part
0130                 ;       into the 'datai' register.
0131                 ;*****************************************************************
0132                 RX
0133   0021  006D          clrf    datai           ; clear input buffer
0134   0022  0C08          movlw   .8              ; set # bits to 8
0135   0023  0031          movwf   count
0136   0024  0403          bcf     status,0        ; make sure carry bit is low
0137   0025  036D  RXLP    rlf     datai           ; rotate the buffer left 1 bit
0138   0026  091A          call    BITIN           ; read 1 bit
0139   0027  040D          bcf     datai,0         ; assume the input bit was low
0140   0028  06EA          btfsc   eeprom,di       ; check the bit
0141   0029  050D          bsf     datai,0         ; set high if necessary
0142   002A  02F1          decfsz  count           ; 8 bits done?
0143   002B  0A25          goto    RXLP            ; no, do another
0144   002C  0800          retlw   0
0145                 ;
0146                 ;*****************************************************************
0147                 ;       BITOUT routine
0148                 ;       This routine takes one bit of data in 'do' and
0149                 ;       transmits it to the serial EE device
0150                 ;*****************************************************************
0151                 BITOUT
0152   002D  07CA          btfss   eeprom,do       ; check state of data bit
0153   002E  0A31          goto    bitlow          ; low, goto bitlow

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:20 1994  Page 4

Line   PC    Opcode

0154   002F  05C6          bsf     port_b,datin    ; high, set datain high
0155   0030  0A32          goto    clkout          ; and clock it
0156                                               ;
0157   0031  04C6  bitlow   bcf     port_b,datin    ; output a logic low
0158   0032  05A6  clkout   bsf     port_b,sclk     ; set clock line high
0159   0033  0000          nop
0160   0034  04A6          bcf     port_b,sclk     ; return clock line low
0161   0035  0800          retlw   0
0162                 ;
0163                 ;*****************************************************************
0164                 ;       Transmit Data Subroutine
0165                 ;       This routine takes the byte of data stored in the
0166                 ;       'datao' register and transmits it to the serial EE device.
0167                 ;*****************************************************************
0168                 TX
0169   0036  0212          movf    bits,w          ; set the number of bits to xmit
0170   0037  0031          movwf   count
0171                       ;
0172                 TXLP
0173   0038  04CA          bcf     eeprom,do       ; assume bit 7 is low
0174   0039  06F0          btfsc   txbuf,7         ; is bit 7 clear?
0175   003A  05CA          bsf     eeprom,do       ; no, set data bit =1
0176   003B  092D          call    BITOUT          ; transmit 1 bit to serial EE
0177   003C  0370          rlf     txbuf           ; rotate txbuf left
0178   003D  02F1          decfsz  count           ; all bits done?
0179   003E  0A38          goto    TXLP            ; no, do another bit
0180   003F  0800          retlw   0               ; yes, jump out
0181                 ;
0182                 ;*****************************************************************
0183                 ;       POWER-UP ROUTINE
0184                 ;       This is the program entry point, which in this case simply
0185                 ;       sets the port_a I/O lines and directs control to the
0186                 ;       erase/write enable routine.
0187                 ;*****************************************************************
0188                 PWRUP
```


© 1994 Microchip Technology Inc.

```
0189                    ;
0190    0040    0C11            movlw   b'00010001'
0191    0041    0005            tris    port_a          ; set RA0 as input, rest output
0192    0042    0065            clrf    port_a          ; all lines low
0193    0043    0C80            movlw   b'10000000'
0194    0044    0006            tris    port_b          ; set RB7 as input, rest output;
0195                    ;
0196                    ;               Fall through and do the read
0197                    ;
0198                    ;*****************************************************************
0199                    ;       READ ROUTINE (Sequential read mode)
0200                    ;       This routine reads 8 consecutive addresses in
0201                    ;       sequential mode starting at address 0.  This
0202                    ;       program will repeat forever
0203                    ;*****************************************************************
0204                    READ

16c5x/7x Cross-Assembler V4.12 Released  Mon Jun 06 10:49:20 1994  Page 5

Line    PC      Opcode

0205                    ;
0206    0045    0C00            movlw   .0              ; set starting address to 00
0207    0046    002C            movwf   addr            ;
0208    0047    0C08            movlw   .8              ; set number of addresses to
0209    0048    0033            movwf   bytcnt          ; read as 8
0210                                                    ;
0211    0049    090E            call    BSTART          ; generate the start bit
0212    004A    0C02            movlw   .2              ; set # bits to 2
0213    004B    0032            movwf   bits            ;
0214    004C    0C80            movlw   b'10000000'     ; get opcode (10b)
0215    004D    0030            movwf   txbuf           ; into output buffer
0216    004E    0936            call    TX              ; and transmit it
0217    004F    0C08            movlw   .8              ;
0218    0050    0032            movwf   bits            ; set number of bits to 8
0219    0051    020C            movf    addr,w          ; get the address
0220    0052    0030            movwf   txbuf           ; into the output buffer
0221    0053    0936            call    TX              ; and transmit it
0222
0223    0054    0921    rbyte   call    RX              ; read the high byte
0224    0055    020D            movf    datai,w         ; move input data to w
0225    0056    0037            movwf   hbyte           ; xfer it to high byte
0226
0227    0057    0921            call    RX              ; read the low byte
0228    0058    020D            movf    datai,w         ; move input data to w
0229    0059    0037            movwf   hbyte           ; xfer it to low byte
0230
0231    005A    02AC            incf    addr            ; add 1 to the address
0232    005B    02F3            decfsz  bytcnt          ; have all bytes been read?
0233    005C    0A54            goto    rbyte           ; no, read another byte
0234
0235    005D    0486            bcf     port_b,chpsel   ; yes,clear the chip select line
0236    005E    0A45            goto    READ            ; and start over
0237            0000    END
```


© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

1.8 Volt Technology - Benefits

Portability is and will continue to be one of the key features driving the design and development of new electronic equipment. Existing products such as palmtop computers, cellular phones and data acquisition devices are becoming increasingly smaller and more powerful. Handheld markets will demand that product feature sets be enhanced and that battery operating life be increased. These market requirements, in turn, will put more demands on battery and integrated circuit technology.

In order to optimize battery technology ICs must be able to operate at voltages well below the 5V range that most ICs operate at today. In fact, even integrated circuits operating at 3 volts do not make the most of the available battery technology. Thus the need for devices that operate below 3 volts.

BATTERY TECHNOLOGY OVERVIEW

Primary versus Secondary

In portable or handheld equipment there are two major battery types employed; primary and secondary. Primary cells are disposable, and alkaline and lithium cells are the main primary batteries in use today. Alkaline batteries are extremely common in consumer applications such as electronic games, cordless phones and palmtop computers. While lithium batteries are used as a primary power source in other consumer products such as cameras and as a back-up supply in palmtops.

Secondary cells are rechargeable batteries. Among the many secondary cells available, lead-acid, Ni-cad and Nickel Metal Hydride (NMH) are the most popular.

Voltage Ratings

Batteries for portable applications, whether primary or secondary, have three different voltage ratings; Rated or nominal, Operating, and End-of-life. The rated voltage is usually the open circuit voltage, which in the case of alkaline "AA" cells is 1.5V. Under normal load the "operating voltage" is realized and in all cases is less than the nominal voltage. The final battery voltage rating is the end-of-life voltage, which is defined as the voltage at which 100% of the usable power of the battery is consumed, or as 75% of the operating voltage. In the case of the same alkaline battery the end-of-life voltage is 0.9 volts. Table 1 lists the operating and end-of-life voltages for a single "AA" or equivalent cell of the five battery types mentioned above.

TABLE 1 - BATTERY VOLTAGE SUMMARY

BATTERY TYPE	OPERATING VOLTAGE	END-OF-LIFE VOLTAGE
ALKALINE	1.2V	0.9V
LITHIUM	2.7V	2.0V
LEAD-ACID	2.0V	1.75V
NI-CAD	1.2V	0.9V
NMH	1.2V	1.0V

With applications employing primary cells this means that batteries are replaced less often, thus reducing operating cost as well as having a positive effect on the environment due to the use and disposal of fewer batteries. In systems with secondary cells time between recharging increased, thus enhancing the performance of the product.

LOW VOLTAGE APPLICATIONS

As mentioned previously, the need for increased portability will drive the employment of low voltage technology. Portable and handheld applications requiring low voltage (sub 3V) technology will be numerous and will in many cases be high volume. These include a variety of personal communications devices such as cellular and cordless phones, computing devices like palmtops and portable PCs, and a number of data acquisition devices for a variety of medical, industrial and commercial applications.

8

© 1993 Microchip Technology Incorporated

1.8 Volt Technology - Benefits

MICROCHIP'S LOW VOLTAGE PRODUCT POSITION

Prior to the announcement of a family of 1.8V Serial EEPROMs in November of 1992, Microchip had already established itself as a leader in low voltage semiconductor products. Microchip's current product portfolio consists of: (1) Serial EEPROMs that operate, both READ and WRITE, down to the following voltage levels: 1.8V, 2.0V, 2.5V and 5V; (2) ROM based microcontrollers that operate down to 2V; and (3) 3V One Time Programmable EPROMs.

The 1.8V Serial EEPROM family operates down to 1.8V without the relaxation of any specification. This includes all AC, DC retention and endurance parameters. Currently the 1.8V family consists of three 3-wire devices, in densities ranging from 1K to 4K bits. Packaging options include 8 pin SOIC and 8 pin PDIP devices. Typical operating currents for these devices, at 1.8V, are in the 70uA range, resulting in power consumption levels of less than 150uW. This compares to 1 to 5 mW for comparable devices operating at 5V. In the near future, the 1.8V family of Serial EEPROMs will be expanded to include all 2-wire devices.

SUMMARY

With the introduction of 1.8V Serial EEPROMs, Microchip has made available silicon technology that optimizes existing battery technology. This over time will result in the development, production, marketing and application of more portable and higher performance handheld and portable computing, telecommunications, and data acquisition products.

Author: *Tom Tyson*
Memory Products Division

© 1993 Microchip Technology Incorporated

MICROCHIP

AN551

Serial EEPROM Solutions vs. Parallel Solutions

In searching for solutions to their system non-volatile memory requirements, equipment, systems and product designers are faced with a plethora of design related issues and trade-offs. The non-volatile memory options available to them offer a variety of different device features including performance, ease of design, power consumption, operating voltage, programmability, density, and physical size. For the most part these non-volatile memory options can be grouped into two major categories: serial solutions and parallel solutions. This paper discusses the attributes of each, conducts a comparative analysis and in the process identifies the benefits and advantages of Serial EEPROMs.

SERIAL EEPROMS

The main feature that makes a device a "Serial" and sets it apart from parallel devices is, as its name implies, the ability to communicate through a serial interface. This ability has numerous benefits. First, serial communication is accomplished with a minimum number of I/O's. Serial EEPROMs require only two to four lines (depending on the hardware and software protocol) for complete communication; memory addressing, data input and output, and device control. Thus, the hardware interface requirements for Serial EEPROMs are kept at a minimum. The most common Serial EEPROMs in use today are devices that utilize a 2-wire protocol.

Another benefit of serial communication is package size. Ranging from densities of 256 to 16K bits, most Serial EEPROMs today are available in space-saving 8 pin PDIP and 150mil wide SOIC packaging. This obviously is very beneficial for applications where product size and weight is a key design factor. The final benefit is low current consumption. Due to a limited number of I/O ports and therefore on-chip support requirements, operating currents for Serial EEPROMs are usually well below 3 milliamperes.

Other features of Serial EEPROMs include: 1) Byte programmability—The ability to erase and program one byte at a time without affecting the contents of the other memory locations in the array; 2) Clock rates of up to 6MHz—2-wire devices are rated at 100KHz and 400KHz per the standard I²C protocol, while 3-wire devices can be operated at 6M Hz rates; 3) Low voltage operation—Microchip has introduced a family of devices that operate, both read and write, down to 1.8V. This family complements other 2V and 2.5V low voltage Serial EEPROM families available.

PARALLEL NON-VOLATILE MEMORIES

There are a number of memory devices that fall into this category. The major ones include Parallel EEPROMs, Flash memory products, EPROMs, and SRAMs with battery back-up.

The main common feature of all of these devices is that communication with the device is done through a parallel interface, which results in a high system clock rate. Each type of device has separate data, address and control lines. Thus pin counts are in the 24 to 40 pin range. This also results in relatively large and costly packages and large footprints, even with the most advanced surface mount packages like TSOP. SRAMs with on-board batteries require DIP package heights that are significantly higher than those of standard DIP packages, adding to its package size and cost disadvantage.

Parallel EEPROM and battery backed-up SRAMs are the only two of the four major types of parallel non-volatile memories that have the capability to erase and program one byte at a time. EPROM and Flash devices require the whole array or at least large sectors to be erased prior to reprogramming.

SERIAL VERSUS PARALLEL

Serial EEPROMs have five major advantages over parallel non-volatile memories.

1) Lower Current Consumption - The maximum operating current (at 5 volts operating voltage) of a 16K serial EEPROM device is approximately an order of magnitude less than that of an equivalent density parallel EEPROM. Operating currents for 16K Serials are specified at 3mA, while 16K parallel devices are specified at 30mA and above. This relationship will continue as 64K serial devices are introduced. Since power consumption is directly proportional to current consumption, the lower the current the lower the power consumption.
2) Lower Voltage - Serial EEPROMs have been available in single supply low voltage options for some time. As mentioned above, Microchip has low voltage Serial EEPROMs that operate down to 1.8V, as well as other low voltage Serials that function down to 2.0V or 2.5V volts. 3V EPROMs

8

© 1993 Microchip Technology Incorporated

and parallel EEPROMs and single voltage 5V flash devices are just being introduced to the market. (Most flash devices on the market today require 12V for programming in addition to the 5 volts required for normal operation). Low voltage operation also has a positive effect on power consumption. A reduction in the operating voltage from 5 volts to 1.8 volts will result in a power consumption reduction of almost 90% and almost a 65% reduction in power if the operating voltage is reduced from 3V to 1.8V.

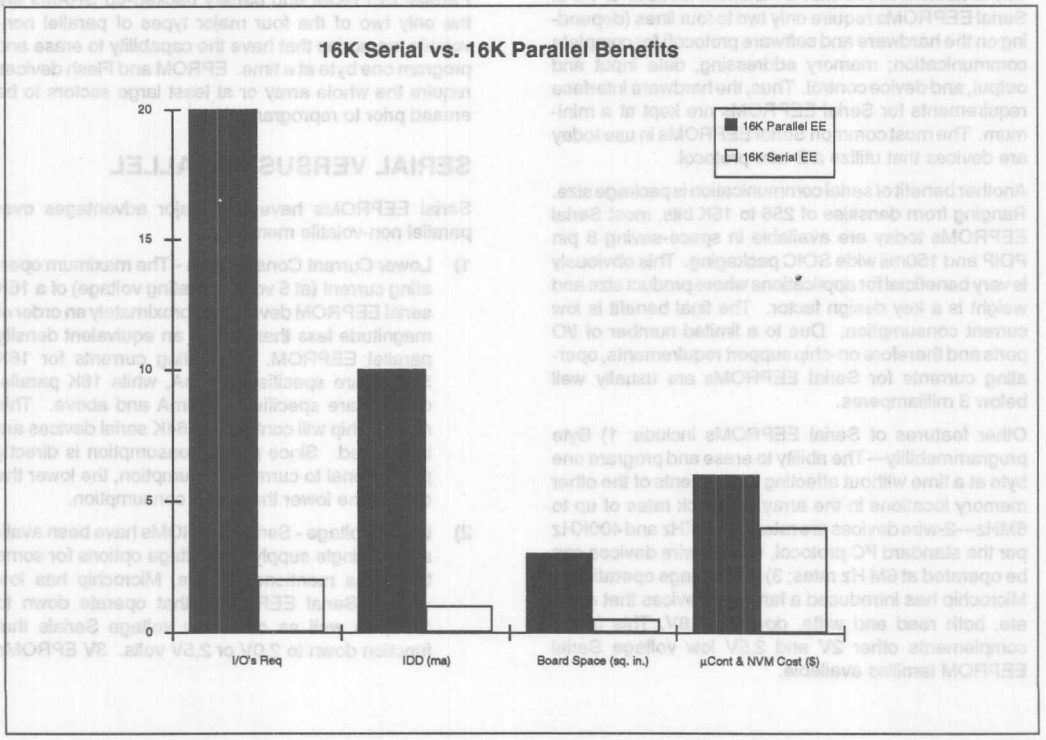
3) Programmability - Neither currently available Flash devices nor EPROMs have the ability to program one byte at a time. Erasing is an array or sector function. Therefore, whenever one byte needs to be reprogrammed the entire array or sector must be reprogrammed. This procedure takes a relatively long amount of time to complete, time which may not be available, as is the case when storing critical parameters or data during inadvertent and unexpected system power loss. This procedure also requires software overhead to manage the retrieval and reprogramming operation.

4) Physical Size - Again, when comparing a 16K Serial EEPROM to a 16K parallel device the serial has a significant advantage. The area of the 150mil 8 pin SOIC footprint is less than 50K square mils. This compares to an area of more than 250K square mils for a 24 pin SOIC and almost 800K square mils for a 24 pin 500mil DIP package footprint.

5) I/O Requirements - Serials only require 2 to 4 input or output lines for complete communications. Most parallel devices require at least 22 lines, depending on the memory density. This results in increased microcontroller/microprocessor overhead and additional real estate to accommodate the numerous hardware lines.

The advantages that parallel devices currently have over serial EEPROMs is memory density and AC performance. However, in most microcontroller based applications for which Serial EEPROMs are intended, high density and AC are not the most critical design issues or most needed product features.

The key benefits of Serial EEPROM solutions as a result of the advantages outlined above, are reduced system costs, enhanced feature sets, and improved system performance. System size and weight is reduced and power sourcing requirements are kept at a minimum. The following graph compares some of the main attributes of a 16K Serial EEPROM device to a 16K Parallel device.



© 1993 Microchip Technology Incorporated

USES AND APPLICATIONS OF SERIAL EEPROMS

Uses of Serial EEPROMs

The days of simply being a DIP switch replacement for Serial EEPROMs is over. Here is a list of the functions that Serial EEPROMs perform in a variety of computer, industrial, telecommunication, automotive and consumer applications:

1) Memory storage of channel selectors or analog controls (volume, tone, etc.)
2) Power down storage and retrieval of events such as fault detection or error diagnostics
3) Electronic real time event or maintenance log such as page counting
4) Configuration storage
5) Last number redial and speed dial storage
6) User in-circuit look-up tables

Serial EEPROM Applications

Serial EEPROMs have found homes in hundreds of embedded control applications in all major application markets. The following list demonstrates the number and variety of applications for serial EEPROMs.

Market	Applications
CONSUMER	TV tuners, VCRs, CD players, cameras, radios, and remote controls
COMPUTER/OA	Printers, copiers, PCs, palmtop and portable computers, disk drives and organizers
INDUSTRIAL	Bar code readers, point-of-sale terminals, smart cards, lock boxes, garage door openers, test measurement equipment and medical equipment
TELECOMM	Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers
AUTOMOTIVE	Air bags, anti-lock brakes, odometers, radios and keyless entry

Using Serial EEPROMs for critical data and configuration storage has only recently become a reality. The current offerings of 2- and 3-wire serial devices offers the systems designer interesting alternatives to the standard parallel EEPROM devices. The Serial EEPROM is basically a standard EEPROM array without the normal parallel data and address I/O. These functions are handled via serial I/O ports coupled with internal self-timed state machines. Not only will the serial device

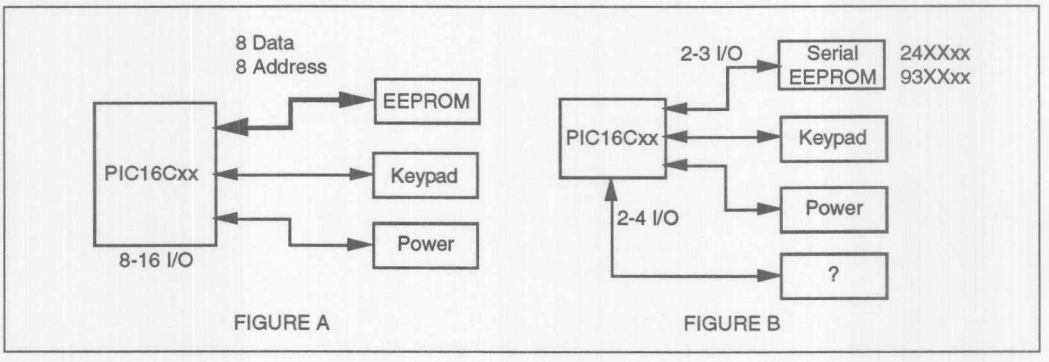


FIGURE A FIGURE B

8

© 1993 Microchip Technology Incorporated

save power, board space, and cost, but they also offer the advantage of fewer I/O and consequently power in the embedded microcontroller because less I/O are needed to control the same functions. A typical embedded application is shown in Figure A, depicting a controller and several functions used in a personal communications device, such as a mobile or portable phone. The EEPROM stores speed dial and last number redial numbers, credit card numbers, ID numbers, and configuration parameters.

Figure B shows these same functions using a controller with fewer I/O and a Serial EEPROM. There is no loss of functionality but a significant savings in current, board space, I/O pads, and cost. The serial solution employs 8 to 16 less I/O on the microcontroller, freeing up much needed functionality, and possibly allowing for a much smaller device package and downsized circuit boards.

SUMMARY

Serial EEPROMs are ideal cost effective solutions to all non-volatile memory embedded control applications that require: 1) A small footprint space saving format; 2) The ability and ease of programming one byte at a time; 3) Low current consumption and low operating voltage; 4) Low microcontroller overhead and support; and, 5) The best price performance non-volatile memory solution available.

Their size, ease of programmability, low power consumption, and low cost make Serial EEPROMs extremely suitable for all the fast growing handheld and portable battery powered computer, personal communications, medical and industrial markets.

Author: *Tom Tyson*
Memory Products Division

© 1993 Microchip Technology Incorporated

Questions and Answers Concerning Serial EEPROMs

Interfacing a serial EEPROM to a controller can be a confusing task, especially for first time users. Because communication with a parallel memory device is more straightforward, the advantages of serial devices are sometimes overlooked because of the time required to learn a different protocol. In order to minimize this time, Microchip provides many application notes on how to interface a controller to our serial memory products. Since every application is different, questions on different aspects of serial memories still arise. This application note provides answers to some of these commonly asked questions on using serial EEPROMs. This series of questions and answers covers a broad range of topics concerning both 2-wire and 3-wire serial EEPROMS, and covers material and/or circumstances that may not be provided in the data book.

QUESTIONS ON 2-WIRE DEVICES (24XXXX OR 85XXXX)

Q: What happens if a STOP bit is inadvertently sent to the device in the middle of a page write command?

A: It depends on which device you are using. The older 24Cxx devices will write any fully loaded bytes in the page to the device. For example if the stop bit occurs before all bits are loaded in the 5th byte, then the first 4 bytes will be written and the 5th byte will not. On the 24LCxxB or 24AAxx devices, if the stop bit occurs before all 8 bits in a byte are loaded, then the entire command will abort and no bytes will be written.

Q: What happens if a START bit is inadvertently sent to the device in the middle of command?

A: A start bit will always reset the part. It should be noted that if the device has control of the SDA line for an acknowledge bit or a read command, it may not be possible to send a start bit.

Q: I'm using interrupts in my controller code, and it is possible that I may jump out of my serial EEPROM communication routine to handle an interrupt. Is there any problem with leaving the device sitting in the middle of a command as long as the clock remains stable?

A: It is perfectly valid to leave a command in the middle indefinitely as long as the clock line remains stable.

Q: I'm using an interrupt routine in my controller code that may leave the serial EEPROM in the middle of a read command. When I return from the interrupt, is there any way to reset the part to make sure it is in a known state before I start sending it commands again?

A: If the part was in the process of outputting data, you can always get it back to standby mode by allowing the SDA line to float high and give it 9 clocks. This assures that the device will not receive the acknowledge bit at the end of the current byte and will abort the command and go to standby.

Q: I'm using a 24LCxxB in my application. What happens if the WP line goes from low (unprotected) to high (protected) in the middle of a write command?

A: The state of the WP pin is OR'ed with the write control circuitry inside the device. The state of the pin is not latched at any point of the command to see if the write should be done or not. Therefore, if the WP pin went high before the stop bit was sent to initiate the command, then the array would be protected. If the WP pin went high *after* the stop bit was sent, then the outcome is unpredictable, because the result would vary according to how far into command the WP pin went high.

Q: I'm attempting to use a serial EEPROM in an application for the first time, and am having problems getting it to work. I have followed all the timing diagrams in the databook but when I attempt to read data from the device I only read zeros.

A: When a read command is initiated and all the data is zeros, the problem is most likely one of two things:

1) A pull-up resistor on the SDA line is required for the device to operate correctly. Common values are 10K - 20K ohms. If this pull-up is not provided, then the device can only output zeros.

2) The controller is not releasing the bus to the device and is holding it low. Check the controller code and make sure that the SDA line has been set for input during the read command.

8

© 1994 Microchip Technology Incorporated

Questions and Answers Concerning Serial EEPROMs

Q: I had been using a 24C04A in my application and have switched to the 24LC04B so I can run at a lower voltage. Now the circuit is no longer working correctly. The read seems to work fine but the write seems to not work at all or only intermittently.

A: The problem is probably with the difference in write cycle times between the two devices. The write cycle time for the 24C04A is about 0.5ms per byte, while the cycle time for the 24LC04B is around 2-4ms per byte. Therefore, when you send consecutive write commands using the new device, it is ignoring all commands that occur while it is in the middle of a write cycle.

Q: I'm using a 24LC16B device in my application. Is there any way I can determine when the write cycle is complete instead of waiting the maximum 10ms?

A: Yes, you can determine when the write cycle is complete on any of our 2-wire devices by utilizing a simple technique called "data polling" or "acknowledge polling." Data polling is done after the stop bit has been sent to the device to initiate a write cycle. You simply send the device a start bit followed by a proper control byte with the R/W bit set low. You then check the status of the acknowledge bit. If the acknowledge bit is high, then the device is still busy writing. You can continue to send the start bit and control byte until the device acknowledges. When the acknowledge bit reads low, then the device has finished the write cycle. You can now send the address and data bytes if you are going to do a write command, or just send the address if you are going to do a read command. Of course you may also send a start bit and begin the next command from scratch.

Q: I'm confused about when the acknowledge bit from the device actually occurs; can you clarify this?

A: The device will attempt to take over the SDA line for the acknowledge bit on the falling edge of the clock for the last bit in the byte. It will then release the SDA line on the falling edge of the clock given for the acknowledge bit. For example, when the control byte is sent to the device, it will take the SDA line low on the falling edge of the clock for the R/W bit. It will then release the SDA line on the next falling edge of the clock (the acknowledge bit). The same sequence occurs for sending the device a byte of data or address.

Q: I am using a 24LC01B in my application which has an 8 byte page buffer. What happens if I load more than 8 bytes into the device before giving it the stop bit?

A: For all 24LCxxB products, the 24C04A and the 85C92, the page buffer will wrap around to the beginning of the buffer and begin to overwrite the data that has previously been loaded For the 24C01A/02A and 85C72/82, the write command will abort if more bytes than the page buffer will hold are loaded.

Q: Can I send a stop bit and a start bit with the same clock pulse?

A: Yes, this is valid as long as setup and hold times for both the start and stop bits are obeyed.

Q: What happens if I give the device a start bit and then a stop bit in the same clock pulse?

A: The 24Cxx devices will ignore the stop bit, the 24LCxxB products will accept the stop bit and go to standby mode.

Q: Does the address pointer get incremented after a current address read command?

A: A current address read command can be halted by sending a stop bit or by sending a high acknowledge bit after the data has been sent by the device. On the 24Cxx devices, ending the read command with a stop bit will increment the address pointer to the next address. Ending the command with a high ack bit will not increment the address.

For the 24LCxxB products, ending the command with either the stop bit or the high ack bit will increment the address pointer.

Q: I am using the 24LC16B and want to write to the part using the page write mode. Can I start a page write at any location in the device?

A: You can start a page write at any location in the device, but you need to be aware that the part is arranged in 16-byte pages and you cannot cross a page boundary. If you attempt a page write that goes beyond the end of the current page, data loaded will 'roll around' to the beginning of the current page instead of going to the next one. For example; the first page in the device starts at address 0x00 and ends at 0x0F. If you start a page write at address 0x0E, the first byte loaded will be written at address 0x0E, the second at 0x0F and the third byte loaded will roll around to the beginning of the page and be written to address 0x00.

© 1994 Microchip Technology Incorporated

QUESTIONS AND ANSWERS FOR 3-WIRE DEVICES (93XXXX)

Q: What would happen if during a write command I inadvertently sent either too many or too few clocks to the device before dropping the CS line?

A: If you send too many clocks to the device and then drop the CS line to initiate the write cycle, the extra clocks will be ignored but the command will execute. If you do not send enough clocks and then drop the CS line, then the command will abort and no write will take place.

Q: I am using a 93LC56 device in my application and am having problems getting it to work correctly. The read sequence seems to work fine but I unable to write any data to the part.

A: A problem such as this is usually caused by either not giving the part the required number of bits for the command before dropping the CS line, or not drop ping the CS line at all. The write command will not commence until the CS line is brought low.

Q: I am currently using a 93C46 device and am looking to go to the 93LC46 device so I can run at a lower voltage. What are the main differences between these devices?

A: There are several operational differences between these devices that you should be aware of:

1) The 93LC46 operates in both x8 and x16 modes; the 94C46 is x16 only.
2) The 93LC46 supports the sequential read function, the 93C46 does not.
3) The write cycle on the 94LC46 begins on the falling edge of the CS line; the write cycle for the 93C46 begins on the rising edge of the last clock.

Q: Is the 93LC46 drop-in compatible with the 93C46 or would I be better off using the 93LC46B?

A: The 93LC46 and 93LC46B are the same device with one exception; the ORG pin on the 93LC46B is internally tied so it will only operate in the x16 mode. This allows the user to leave pin 7 floating, just like you would on the 93C46. For this reason, the 93LC46B is a closer match than the 93LC46. Please note the other operational differences described in the previous question.

Q: I am confused as to whether I have to toggle the CS line low in-between every command.

A: Yes, the CS line must go low for at least 250 ns between each command. If you are doing a write command and you bring CS low to activate the data polling mode, you must toggle CS low *again* after the ready signal has been given by the device before the next start bit can be sent.

Author: *Bruce Negley*
Memory Products Division

8

© 1994 Microchip Technology Incorporated

NOTES:

© 1994 Microchip Technology Incorporated

SECTION 9
DEVELOPMENT TOOLS

Development System Selection Chart 9- 1
Microchip Bulletin Board Service (BBS) 9- 3
Systems Information Line 9- 5
Application Specific Standard Products Tools:
PICSEE™ Product Development Tools 9- 7
TrueGauge™ Intelligent Battery Management Development Tool 9- 11
Logic Product Tools:
PICMASTER™ Product Brief 9- 17
PRO MATE™ Product Brief 9- 21
PICSTART™-16B1 Product Brief 9- 25
PICSTART™-16C Product Brief 9- 27
PICDEM-1 Product Brief 9- 29
PICDEM-2 Product Brief 9- 31
MPASM Universal Assembler Product Brief 9- 33
MPSIM Simulator Product Brief 9- 35
MP-C Product Brief 9- 37
*fuzzy*Tech®-MP Product Brief 9- 39
Memory Products Tools:
Total Endurance™ Serial EEPROM Endurance Model 9- 41
Designer's Kit 9- 43

9

© 1994 Microchip Technology Inc.

© 1994 Microchip Technology

MICROCHIP

Development System Selection Chart

	PIC16/17 Microcontroller											ASSP			Memory	
PRODUCT	PIC16C54	PIC16C54A	PIC16C55	PIC16C56	PIC16C57	PIC16C58A	PIC16C64	PIC16C71	PIC16C74	PIC16C84	PIC17C42	MTA81010	MTA85XXX	MTA11200	Serial EEPROM	Page
Emulator Systems																
PICMASTER™-16B								x								9-17
PICMASTER-16C										x						9-17
PICMASTER-16D	x	x	x	x	x	x						†	†			9-17
PICMASTER-16E							x									9-17
PICMASTER-16F									x							9-17
PICMASTER-17											x					9-17
Development Kits																
*fuzzy*TECH®-MP	x	x	x	x	x	x	x	x	x	x	x					9-39
PICSTART™-16B1	x	x	x	x	x	x		x		x						9-25
PICSTART-16C							x		x							9-27
PICSEEKIT™-81A												x				9-7
PICSEESTART™-81A												x				9-8
PICSEEKIT-85A													x			9-9
PICSEESTART-85A													x			9-10
PRO MATE™	x	x	x	x	x	x	x	x	x	x	x	x	x			9-21
Serial EEPROM Designer's Kit															x	9-43
TrueGauge™														x		9-11
Emulator Probe Kits																
PICPROBE-16B								x								
PICPROBE-16C										x						
PICPROBE-16D	x	x	x	x	x	x						†	†			
PICPROBE-16E							x									
PICPROBE-16F									x							
PICPROBE-17A											x					
Demonstration Boards																
PICDEM-1	x	x	x	x	x	x		x		x	x					9-29
PICDEM-2							x		x							9-31

† This probe may be used in the development of the PIC16C5X software for this PICSEE device.

PICMASTER, PICSTART, PICSEE, PRO MATE and TrueGauge are trademarks of Microchip Technology Inc.
*fuzzy*TECH is a registered trademark of INFORM Software Corp.

9

© 1994 Microchip Technology Inc.

Development System Selection Chart

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP BBS

Microchip Bulletin Board Service

Get current information and help on Microchip's Bulletin Board Service (BBS)! Microchip wants to provide you with the best responsive service possible. To accomplish this, the systems team monitors the BBS, posting the latest component data and software tool updates, providing technical help and embedded systems insights, and discussing how Microchip products provide project solutions. Extend your technical groups staff with microcontroller and memory experts through Microchip's BBS communication channel.

CONNECTING TO MICROCHIP

Connect worldwide to the Microchip BBS using the CompuServe® communications network. In most cases, a local call is your only expense. The Microchip BBS connection does not use CompuServe membership services, therefore **you do not need CompuServe membership to join Microchip's BBS.**

The procedure to connect will vary slightly from country to country. Please check with your local CompuServe agent for details if you have a problem. CompuServe services allows multiple users at baud rates up to 9600. To connect:

1. Set your modem to 8-bit, No parity, and One stop (8N1). This is not the normal CompuServe setting which is 7E1.
2. Dial your local CompuServe phone number.
3. Depress **<RETURN>** and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type **+**, depress **<RETURN>** and `Host Name:` will appear.
5. Type **MCHIPBBS**, depress **<RETURN>** and you connect to the Microchip BBS.

In the United States, to find CompuServe's phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600 baud connection. After the system responds with Host Name:, type

NETWORK, depress **<RETURN>** and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 457-1500 for your local CompuServe number.

Connect without charge to the bulletin board. However, you are responsible for your phone charges. Access is available to all, but users are required to register the first time they "log in." No registration fees are required at this time.

USING THE BULLETIN BOARD

The Microchip Bulletin Board is a multi-faceted tool. Topic information includes:

- Special Interest Groups
- Files
- Mail
- Bug lists
- Technical assistance
- Consultant Directory

Special Interest Groups

Special Interest Groups, or SIGs, offer you the opportunity to discuss technical issues and topics with other users. Take advantage of the Microchip user community's broad background to glean information not available by any other method.

SIGs exists for most Microchip systems, including:

- PIC16/17-SW
- PICMASTER™
- PRO MATE™
- UTILITIES
- ENDURANCE
- MEMORY PRODUCTS
- BUGS
- APP NOTE

These groups are moderated by Microchip staff.

Files

The Microchip Systems BBS is used regularly to distribute bug reports, history files, and interim patches for Microchip software products.

Users can contribute files for distribution on the BBS. These files will be monitored, scanned, and approved or disapproved by the SIG moderator. No executable files are accepted from the user community in general.

Mail

The BBS can be used to distribute mail to other users of the service.

This is an excellent way to get questions answered by the Microchip staff, as well as to keep in touch with fellow Microchip product users worldwide.

The BBS is an evolving product intended to serve your needs. We welcome your ideas and input. Consider mailing a message to your SIG moderator, or to the SYSOP, if you have ideas or questions about particular Microchip products, or BBS operation.

CompuServe is a registered trademark of CompuServe Inc.

9

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

Systems Information Hotline

INTRODUCTION

The Microchip Systems Information Hotline provides callers with listing of the latest versions of all Microchip Development System Software products plus information on how to receive any currently available upgrade kits.

Microchip endeavors to ship the latest software available at the time of product shipment, however, we are continually updating and enhancing these products. This information line provides a quick-check method of insuring that you have the very latest software available. The line is updated just as soon as a new software change/update is made. If you find that the software version that you have has been revised, the hotline provides information on how to download the latest versions from Microchip's BBS.

In the event that Microchip has revised a non-software systems product, the information line will provide details about the change plus provide details on how you can receive the upgrade kit.

CONNECTING TO THE SYSTEMS INFORMATION HOTLINE

Connecting to the Systems Information Hotline is easy. If you are calling from most of North America, call 1-800-755-2345. For the rest of the world, call 1-602-786-7302.

9

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

PICSEE™ TOOLS

PICSEE Product Development Tools

INTRODUCTION

The PICSEE Development Systems provide the product development engineer with cost effective and timely design tool solutions for the MTA8XXXX family of 8-bit CMOS microcontrollers with serial EEPROM. They are designed specifically for the MTA8XXXX family. These tools work in conjunction with existing hardware and software design tools for the PIC16CXX microcontroller family. This allows the development engineer to efficiently implement systems utilizing these multichip modules with a minimal learning curve and capital investment.

PICSEEKIT™-81A — P/N AC812001

- Supports MTA81010
- Programming Adapters for PDIP and SOIC packages
- Daughter card for PICPROBE-16A
- I²C™ bus Serial Communication Application Software

This kit supports the MTA81010 multichip module. It contains programming adapters, a PICMASTER™ emulator daughter board and MTA81010 product samples in 28-lead PDIP. Also included is an MS-DOS®, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

Two programming adapters are provided to allow the MTA81010's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA81010's internal EPROM. Also, any programmer that supports Microchip's 24LC01B Serial EEPROM can program the MTA81010's internal Serial EEPROM. There is one adapter for MTA81010's in DIP packages and another for SOIC packages. Both DIP and SOIC programming adapters interface to programmers via a 300 mil DIP header.

FIGURE 1 - PICSEEKIT-81A INTRODUCTION DESIGN KIT



Description of Contents

1 • PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2 • Header Interface for PICMASTER-16D and PICPROBE-16D
3 • Serial EEPROM Example Software Disk
4 • MTA81010 Product Samples
5 • 8- and 18-Pin Programming Adapter Plugs
6 • Complete Systems Documentation

I²C is a trademark of Philips.
MS-DOS is a registered trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

Emulation Support

The emulator daughter board allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA81010 Microcontroller with Serial EEPROM. This daughter board replaces Microchip's PIC16C5X Emulator Probe Header (P/N AC162009) emulator probe to support the MTA81010. The daughter board provides the required translation from a PIC16C54 pin out to the MTA81010 pin out. It also contains a discrete 24LC01B Serial EEPROM to provide the same functions as the MTA81010's internal EEPROM. This provides a cost-effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a serial EEPROM is included in the kit. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART-81A — P/N DV813001

- Complete Low-Cost Development Solution for MTA81010
- Combines PICSEEKIT AC812001 and PICSTART-16B1 DV163003
- MPALC Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC812001) with a PICSTART™-16B1 (P/N DV163003) to form a complete low- cost development system for the MTA81010 multichip module. It is designed to support the MTA81010 during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA81010's internal EPROM program memory. For a more detailed description, please refer to the PICSEEKIT P/N AC812001 and PICSTART P/N DV163003 product descriptions.

FIGURE 2 - PICSEESTART™-81A DEVELOPMENT KIT



Description of Contents

1 • PICSEE PDIP and SOIC to PIC16C54 or 24LC01B Programming Adapter Sockets
2 • Header Interface for PICMASTER-16D and PICPROBE-16D
3 • Serial EEPROM Example Software Disk
4 • MTA81010/PIC16CXX Product Samples
5 • PIC16CXX Device Programmer Board
6 • PIC16CXX Assembler, Simulator and Host Software
7 • 8- and 18-Pin Programming Adapter Plug
8 • Power Supply
9 • RS-232 Cable
10 • Complete Systems Documentation

Preliminary
© 1994 Microchip Technology Inc.

PICSEEKIT-85A — P/N AC852001

- Supports MTA85XXX
- Programming Adapter for SSOP package
- PICMASTER Adapter Socket
- I²C™ bus Serial Communication Application Software

This kit supports the MTA85XXX multichip module. It contains a programming adapter, a PICMASTER adapter socket, and product samples. Also included is an MS-DOS, PC-compatible 3.5-inch software diskette that contains example source code for implementing the I²C serial bus protocol to communicate with a Serial EEPROM. Documentation is provided for all of the included hardware and software.

Programming Support

A programming adapter socket is provided to allow the MTA85XXX's internal program EPROM as well as its data EEPROM to be programmed on existing programmers. Any programmer that supports Microchip's PIC16C54 can program the MTA85XXX's internal EPROM. Also, any programmer that supports Microchip's 24LC01B and 24LC02B Serial EEPROM can program the MTA85XXX's internal Serial EEPROM. There is one adapter for the SSOP package for MTA85XXX.

FIGURE 3 - PICSEEKIT-85A DEVELOPMENT KIT



Description of Contents

1 • 20-Lead SSOP Programming Adapter Socket
2 • PICMASTER Adapter Socket
3 • Serial EEPROM Communications Software Disk
4 • Product Samples
5 • 8- and 18-pin Programming Adapter Plugs
6 • Complete Systems Documentation

9

© 1994 Microchip Technology Inc.

Emulation Support

An adapter socket allows the developer to use Microchip's PICMASTER in-circuit emulator to emulate the MTA85XXX Microcontroller with Serial EEPROM. The adapter socket provides the required translation from a PIC16C54A/58A pin out to the MTA85XXX pin out. It also contains a discrete 24LC02B Serial EEPROM to provide the same functions as the MTA85XXX's internal EEPROM. This provides a cost- effective emulation solution to customers who may wish to purchase a PICMASTER in-circuit emulator or those that already have a PICMASTER.

Software Support

Example source code for I²C Bus communication with a serial EEPROM is included in the kit. This pre-tested code can be used directly or modified by the developer to meet their specific needs. This example code is provided royalty free and license free.

PICSEESTART-85A — P/N DV853001

- Complete Low-Cost Development Solution for MTA85XXX
- Combines PICSEEKIT-85A AC852001 and PICSTART-16B1 DV163003
- MPALC Assembler
- MPSIM Simulator
- Low-Cost Programmer
- Programming Adapter Sockets
- I²C Bus Applications Software

This kit combines the PICSEEKIT (P/N AC852001) with a PICSTART-16B1 (P/N DV163003) to form a complete low- cost development system for the MTA85XXX multichip module. It is designed to support the MTA85XXX during the software development and initial prototype phases of new product development. It contains tools for software development and debugging, as well as programmer for programming the MTA85XXX's internal EPROM program memory. For a more detailed description, please refer to the PICSEEKIT-85A P/N AC852001 and PICSTART-16B1 P/N DV163003 product descriptions.

FIGURE 4 - PICSEESTART-85A DEVELOPMENT KIT



Description of Contents

1 • 20-lead SSOP Programming Adapter Socket
2 • PICMASTER Adapter Socket
3 • Serial EEPROM Communications Sofware Disk
4 • Product Samples
5 • PIC16CXX Device Programmer Board
6 • PIC16/17 Assembler, Simulator, and Host Software
7 • 8- and 18-pin Programming Adapter Plugs
8 • Power Supply
9 • RS-232 Cable
10 • Complete Systems Documentation

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
AC812001	PICSEEKIT-81A FOR MTA81010
DV813001	PICSEESTART-81A FOR MTA81010
AC814001	PDIP PROGRAMMING SOCKET ONLY
AC814002	SOIC PROGRAMMING SOCKET ONLY
AC852001	PICSEEKIT-85A FOR MTA85XXX
AC854001	20-LEAD SSOP PROGRAMMING ADAPTER SOCKET AND ADAPTER PLUGS ONLY
DV853001	PICSEESTART-85A FOR MTA85XXX

© 1994 Microchip Technology Inc.

TRUEGAUGE™

TrueGauge™ Intelligent Battery Management Development Tool

INTRODUCTION

The MTA11200 TrueGauge Intelligent Battery Management IC is supported by a user friendly tool for system development. The DV114001 operates under Microsoft Windows®. This development tool enables for management of all phases of product development, including inception, debugging and maintenance. System design verification can be accomplished before a hardware prototype needs to be built, thus reducing time and cost. The user interface provides a graphically-oriented development environment. The data logging feature saves measured data into a file that can be imported to Excel®.

This comprehensive design package allows the user to input parameters of a system design, and the TrueGauge system analysis show visually how the design will operate. Adjustments can be made to the design as needed. The critical process verifies a system design before a hardware prototype is built. A calibration program is included. Please refer to Application Note AN570, "Calibrating the MTA11200", for details.

FIGURE 1 - TRUEGAUGE DEVELOPMENT TOOL KIT



SUMMARY OF FEATURES

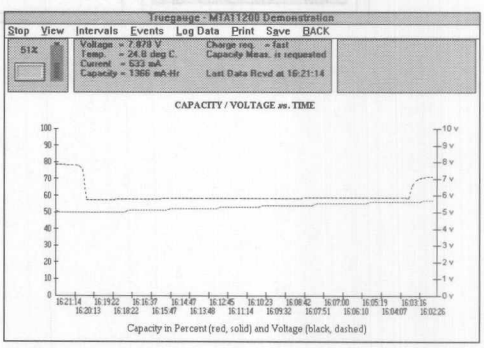
The TrueGauge development tool is a tool for system development under Windows. The development tool kit contains the following:

- NiCd battery with TrueGauge module
- NiMH battery with TrueGauge module
- Stand-alone TrueGauge module
- Charger/Discharger Interface Board
- Universal Power Supply with power cord
- PC Interface Cable with DB9-DB25 converter
- Design/Verification software on a 3.5" diskette
- MTA11200 and 24LC01B product samples
- MTA11200 Datasheet
- TrueGauge Development Tool User's Guide

Battery status information is plotted on the computer allowing for the real-time monitor of battery management parameters including (see Figure 2):

- battery capacity
- voltage
- temperature
- charge/discharge current
- number of charge cycles
- battery errors

FIGURE 2 - CAPACITY/VOLTAGE OVER TIME



TrueGauge is a trademark of Microchip Technology Inc.
Microsoft Windows and Excel are registered trademarks of Microsoft Corporation.

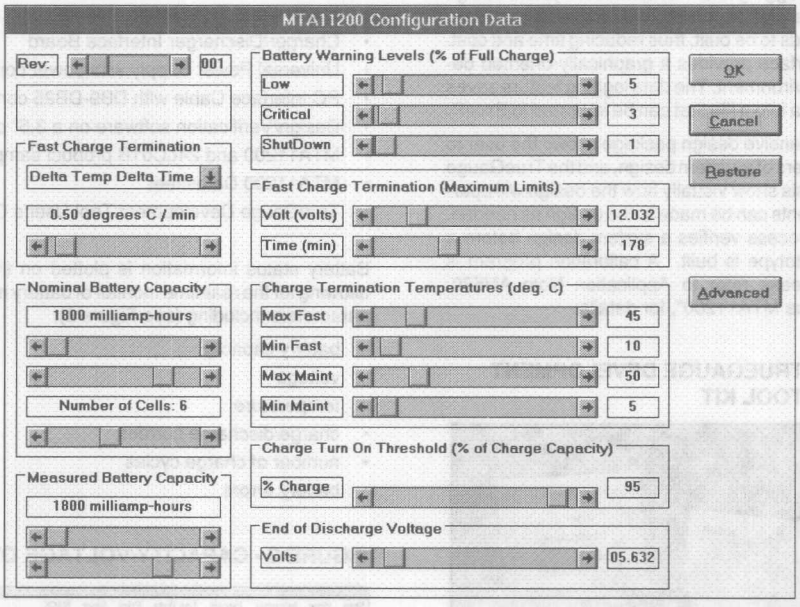
© 1994 Microchip Technology Inc.

Parameters can be changed easily using configuration displays (see Figures 3 and 4) and downloaded to the TrueGauge module using the download command.

Parameters that can be changed are:

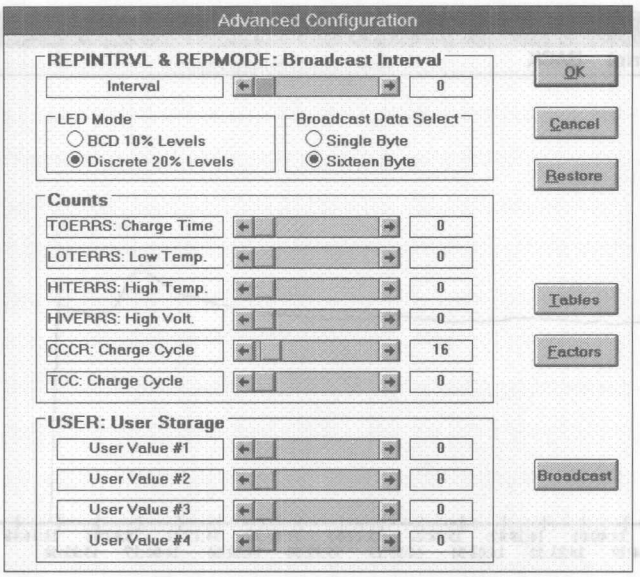
- Fast charge termination
  - Negative Delta Voltage
  - Delta Temp / Delta Time
  - Peak voltage
- Temperature limits for maintenance charge and fast charge
- Battery warning levels
- Communications to host
- LED configuration
- Battery conditioning
- User-defined storage
- Compensation factors for self-discharge capacity and charge efficiency (see Figures 8, 9 and 10)
- End of Discharge (EODV)
- Charge turn-on threshold

FIGURE 3 - CONFIGURATION CONTROL PANEL



© 1994 Microchip Technology Inc.

FIGURE 4 - ADVANCED CONFIGURATION DISPLAY

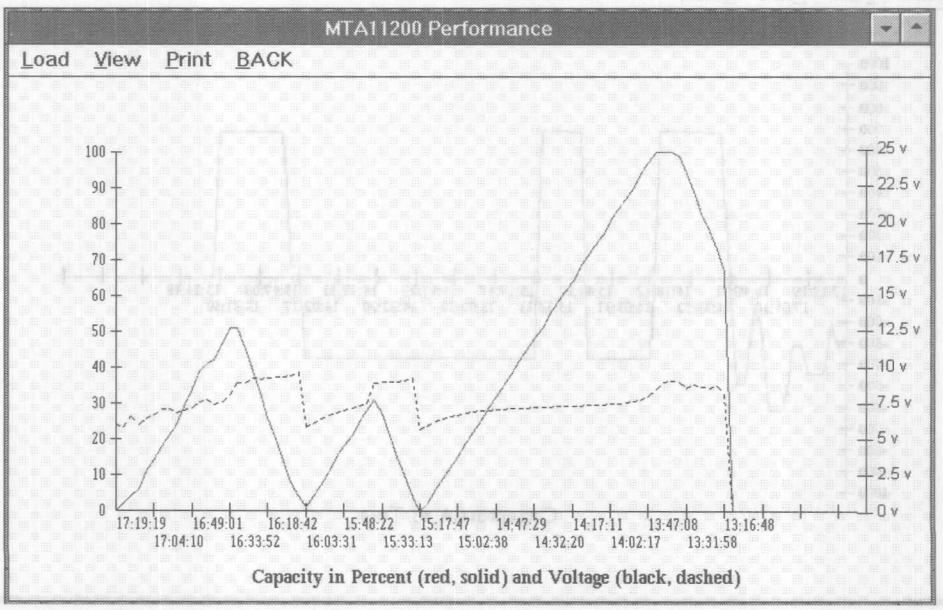


System design verification can be accomplished before hardware implementation by plotting real-time battery activity and performance (see Figures 5, 6 and 7).

The development tool delivers the most comprehensive information, including capacity, voltage, temperature, current and battery status, the number of charge cycles the battery has undergone, and existence of temperature extremes.

TrueGauge supports multiple battery technologies: nickel cadmium, nickel-metal hydride and lead acid battery packs.

FIGURE 5 - TRUEGAUGE VOLTAGE AND CAPACITY VS. TIME



9

© 1994 Microchip Technology Inc.

FIGURE 6 - TEMPERATURE VS. TIME

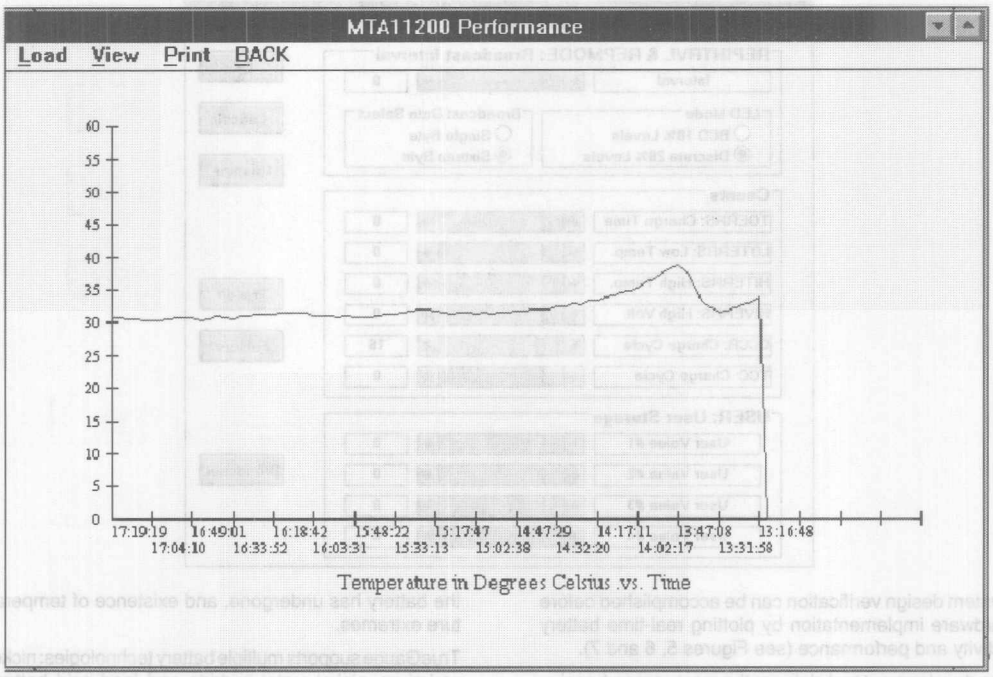
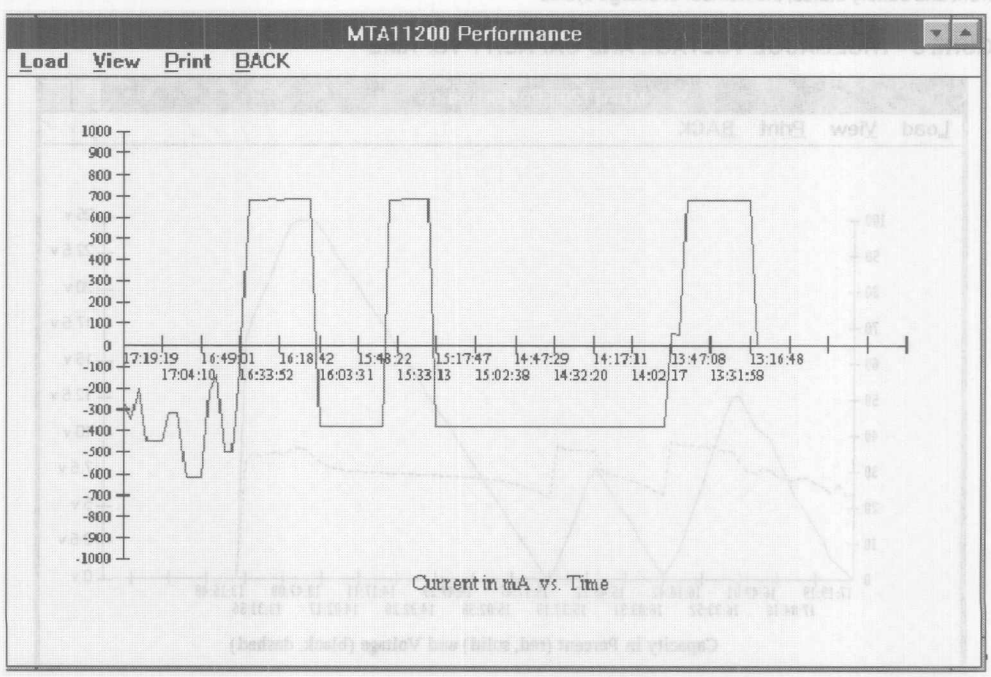


FIGURE 7 - CURRENT VS. TIME



© 1994 Microchip Technology Inc.

FIGURE 8 - CHARGE EFFICIENCY COMPENSATION

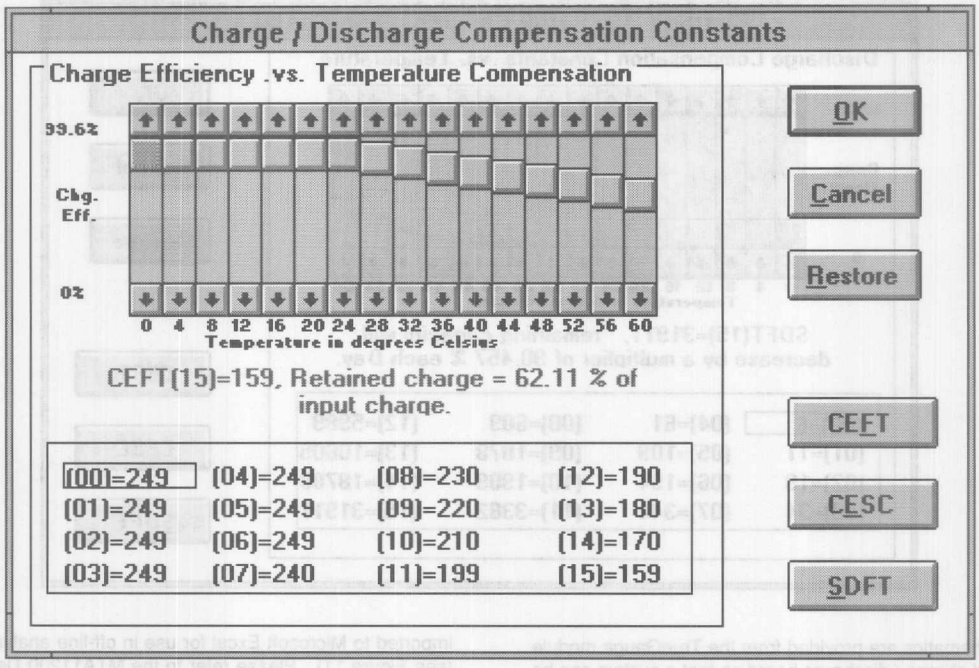
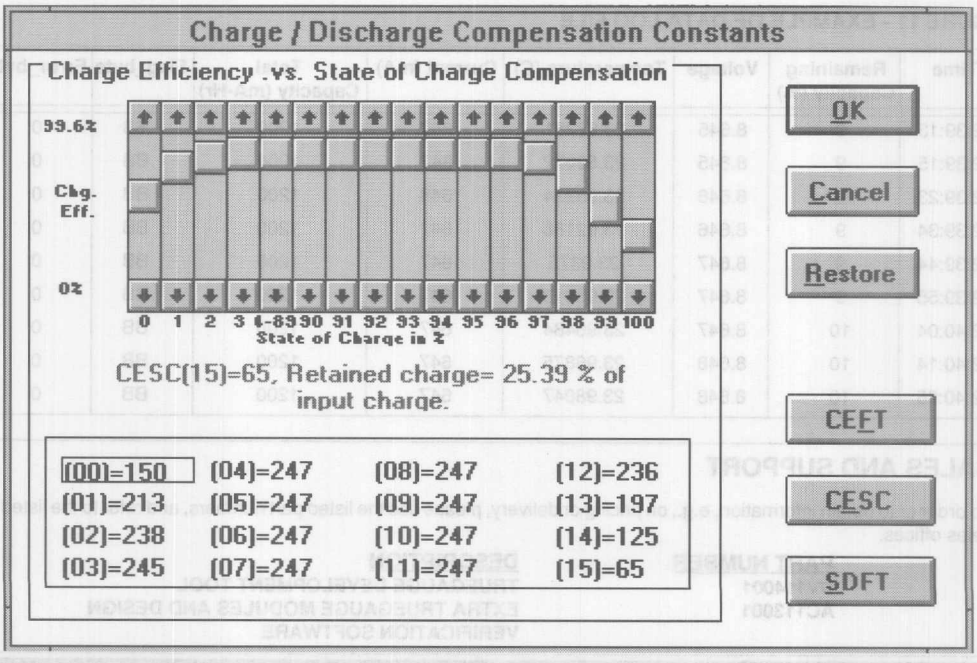


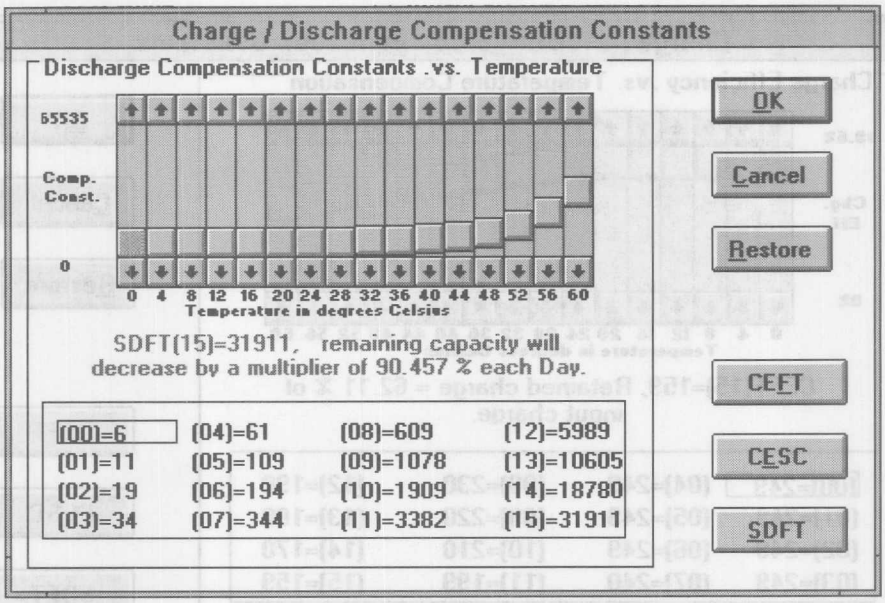
FIGURE 9 - CHARGE EFFICIENCY COMPENSATION (CONT)



9

© 1994 Microchip Technology Inc.

FIGURE 10 - SELF- DISCHARGE VS. TEMPERATURE COMPENSATION



Schematics are provided from the TrueGauge module and charge / discharge board so that a system can be tailored easily to your requirements. A data logging feature saves measured data into a file that can be imported to Microsoft Excel for use in off-line analysis (see Figure 11). Please refer to the MTA11200 Data Sheet for further details on the parameters and the data log file.

FIGURE 11 - EXAMPLE OF DATA LOG FILE

Time	Remaining Capacity (%)	Voltage	Temperature (C)	Current (mA)	Total Capacity (mA-Hr)	Flag_byte	Error_byte
22:39:13	9	8.645	23.88281	647	1200	BB	0
22:39:15	9	8.645	23.88672	647	1200	BB	0
22:39:23	9	8.646	23.90234	648	1200	BB	0
22:39:34	9	8.646	23.92188	647	1200	BB	0
22:39:44	9	8.647	23.9375	647	1200	BB	0
22:39:55	9	8.647	23.95313	647	1200	BB	0
22:40:04	10	8.647	23.96484	647	1200	BB	0
22:40:14	10	8.648	23.96875	647	1200	BB	0
22:40:25	10	8.648	23.98047	647	1200	BB	0

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DV114001	TRUEGAUGE DEVELOPMENT TOOL
AC113001	EXTRA TRUEGAUGE MODULES AND DESIGN VERIFICATION SOFTWARE

© 1994 Microchip Technology Inc.

MICROCHIP PICMASTER™ SYSTEM

PICMASTER Universal In-Circuit Emulator System



SYSTEM FEATURES

General:

- Complete Hi-Performance PC-based Microcontroller Development System for the PIC16/17 families.
- For use on PC-compatible 286, 386, and 486 machines under Microsoft® Windows™ 3.X environment.
- Assembler Software, Emulator System, and EPROM Programmer unit, sample kit, and demonstration hardware and software provide a complete microcontroller product development environment.

Emulator System:

- Hi-Performance In-Circuit Emulation of Microchip Microcontrollers.
- Real-time instruction emulation.
- Single and Multiple instruction step execution.
- Program Memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory.
- Real-time trace memory capture of 40 bits of information for each instruction cycle in an 8Kx40 trace buffer. Trace region can range from 0 to 64K in any address combinations.
- Real-time trace data can be captured and displayed without halting emulation.
- Unlimited number of hardware breakpoints can be set anywhere in the program memory.
- External Break with "AND"/"OR" capability with internal breakpoints.
- Multiprocessor emulation capability. Up to eight PICMASTER emulators can be synchronized on a single PC, for multi-processor development.
- Extended 48-bit cycle counter.
- Trigger Output available on any range of addresses.
- Full Symbolic Debug Capability. Symbolic display and alter of all register files, special purpose registers, stack registers, and bank registers.
- Selectable Internal Emulator Clock or User Target (Prototype) System Clock.
- User selectable internal or external Power Supply (provided).

9

PICMASTER is a trademark of Microchip Technology Inc.
Windows is a trademark of Microsoft Corp.
Microsoft is a registered trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

EPROM Programmer System:

- PRO MATE™ Device Programmer unit for all current PIC16/17 products.
- Operates as a Stand-alone Unit or in Conjunction with a PC-compatible host system.
- Performs READ, PROGRAM, and VERIFY functions in Stand-alone mode.
- PC Host Software provides file display and editing, file transfer to and from programmer unit, device serialization, and program voltage calibration.

Macro Assembler:

- Provides translation of Assembler source code to object code for the PIC16/17 family of microcontrollers.
- Macro-assembly and conditional assembly capability.
- Produces Object files, Listing files, Symbol files, and special files required for symbolic debug with the PICMASTER Emulator System.
- Binary / Hex output formats: INHX8S, INHX8M, INHX16, and PICMASTER.

Demo Board:

The PICDEM-I Demostration Board provides a user with a simple hardware tool through which software can be exercised and debugged. A step-by-step tutorial enables first-time users of PICMASTER to become familiar with all the features of the emulator. A generous prototype area (200 holes) allows the user to build additional hardware for their project.

SYSTEM DESCRIPTION

The PICMASTER Universal In-Circuit Emulator System is intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PIC16/17 family. The PICMASTER system currently supports the PIC16C54, PIC16C55, PIC16C56, PIC16C57 and PIC16C58A at clock frequencies of 20 MHz; the PIC16C64, PIC16C71, PIC16C74, PIC16C84 to 10 MHz; and the PIC17C42 at 16 MHz.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

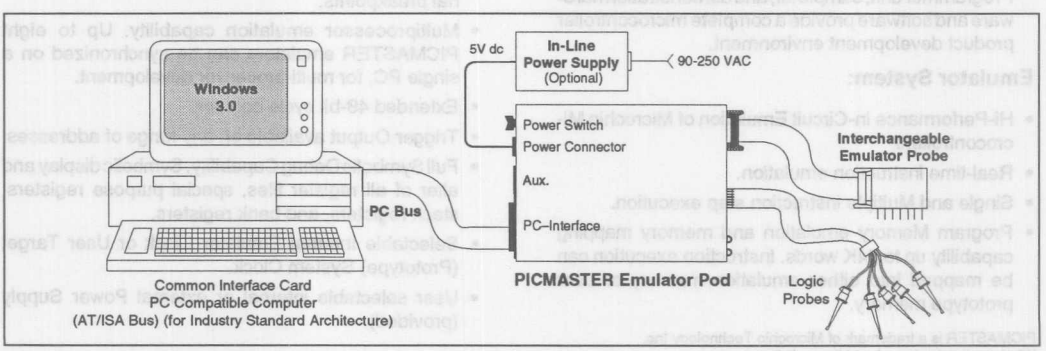
The Emulator System is designed to operate on low-cost PC-compatible machines ranging from 286-AT® class ISA-bus systems through the new 486 EISA-bus machines. The development software runs in the Microsoft Windows 3.X environment, allowing the operator access to a wide range of supporting software and accessories.

Provided with the PICMASTER System is a high performance, real-time In-Circuit Emulator, a microcontroller programmer unit, a macro assembler program, and a simulator program. Sample programs are provided to help quickly familiarize the user with the development system and the PIC16/17 microcontroller families.

Coupled with the user's choice of text editor, the system is ready for development of products containing any of Microchip's microcontroller products.

A "Quick Start" Product Sample Pak containing user programmable parts is included for additional convenience (only devices supported by the specified probe header).

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS system.



PRO MATE is a trademark of Microchip Technology Inc.
AT is a registered trademark of IBM Corp.

© 1994 Microchip Technology Inc.

Host System Requirements:

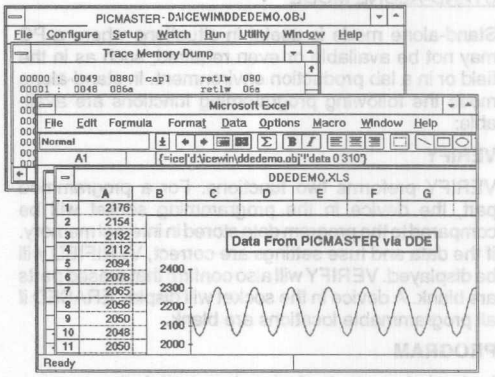
The PICMASTER has been designed as a real-time emulation system with advanced features generally found on more expensive development tools. The AT platform and Windows 3.X environment was chosen to best make these features available to you the end user. To properly take advantages of these features, PICMASTER requires installation on a system having the following minimum configuration:

- PC/AT-compatible machine: 286, 386SX, 386DX, or 486 with ISA or EISA Bus.
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended).
- MS-DOS® / PC-DOS version 3.1 or greater.
- Microsoft® Windows version 3.0 or greater operating in either standard or 386 enhanced mode).
- 1 Mbyte RAM (2 Mbytes recommended).
- One 3.5" floppy disk drive.
- Approximately 10 Mbytes of hard disk (1 Mbyte required for PICMASTER, remainder for Windows 3.X system).
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)
- Microsoft mouse or compatible (highly recommended).

Emulator System Components:

The PICMASTER Emulator Universal System consists primarily of four major components:

- **Host-Interface Card:** The PC Host Interface Card connects the emulator system to a PC compatible system. This high-speed parallel interface requires a single half-size standard AT / ISA slot in the host system. A 37-conductor cable connects the interface card to the external Emulator Control Pod.
- **Emulator Control Pod:** The Emulator Control Pod contains all emulation and control logic common to all microcontroller devices. Emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic are contained here. The Pod controls and interfaces to an interchangeable target-specific emulator probe via a 14" precision ribbon cable.
- **Target-specific Emulator Probe:** A probe specific to microcontroller family to be emulated is installed on the ribbon cable coming from the control pod. This probe configures the universal system for emulation of a specific microcontroller. Currently, the PIC16C5X family, PIC16CXX family, and the PIC17C42 microcontrollers are supported. Future microcontroller probes will be available as they are released.



- **PC Host Emulation Control Software:** Host software necessary to control and provide a working user interface is the last major component of the system. The emulation software runs in the Windows 3.X environment, and provides the user with full display, alter, and control of the system under emulation. The Control Software is also universal to all microcontroller families.

  The Windows 3.X System is a multitasking operating system which will allows the developer to take full advantage of the many powerful features and functions of the PICMASTER system.

  PICMASTER emulation can operate in one window, while a text editor is running in a second window. Dynamic Data Exchange (DDE), a feature of Windows 3.X, will be available in this and future versions of the software. DDE allows data to be dynamically transferred between two or more Windows programs. With this feature, data collected with PICMASTER can be automatically transferred to a spreadsheet or database program for further analysis.

  Under Windows 3.X, up to eight PICMASTER emulators can run simultaneously on the same PC making development of multi-microcontroller systems possible (e.g., a system containing a PIC16CXX processor and a PIC17CXX processor).

PRO MATE Device Programmer:

The PRO MATE Programmer system included in the PICMASTER Development System provides the product developer with the ability to program (transfer) the developer's software into PIC16/17 microcontrollers.

The programmer unit comes complete with accessories for use with a PC host computer. Supplied are interface cables and connectors to a standard PC serial port , a power supply unit, and host operating software.

The PRO MATE Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.

MS-DOS is a registered trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY preforms two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

When the PRO MATE is connected to a host PC system, many more options and conveniences are available to the user. Host mode allows full interactive control over the PRO MATE unit. A full screen, user-friendly software program is provided to fully assist the user.

As in stand-alone mode, parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified. In addition, other features available in host-mode are:

Editing

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program and fuse data can be loaded and saved to DOS disk files. Files generated by the Assembler program are directly loadable into programmer memory.

VDD and VPP Adjust

The programming environment voltage settings of VDD max, VDD min, and VPP can be set and altered only on PC host mode. The voltage settings allow the user to program the part in the environment that the part will be used. The part will be programmed at VDD max and verified at VDD min. VPP is the programming voltage.

PICMASTER PROBE Specifications

Table 1 shows the current probe specifications for the PICMASTER In-Circuit Emulator. The devices are supported regardless of program memory type (ROM, EPROM or EEPROM), process technology or voltage range. That is, selecting the PROBE that supports the PIC16C54 (-16A) also supports the PIC16CR54, PIC16C54A and the PIC16LC54A devices. The probe would also support other variations as they become available (such as PIC16CR54A).

TABLE 1: PICMASTER PROBE SPECIFICATIONS

PICMASTER PROBE	Devices Supported	PROBE Maximum Frequency	PROBE Operating Voltage
PROBE - 16B	PIC16C71	10 MHZ	4.5V - 5.5V
PROBE - 16C	PIC16C84	10 MHZ	4.5V - 5.5V
PROBE - 16D	PIC16C54, PIC16C55, PIC16C56, PIC16C57, and PIC16C58A	20 MHz	4.5V - 5.5V
PROBE - 16E	PIC16C64	10 MHZ	4.5V - 5.5V
PROBE - 16F	PIC16C74	10 MHZ	4.5V - 5.5V
PROBE - 17A	PIC17C42	16 MHZ	4.5V - 5.5V

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the factory or the listed sales offices.

PART NUMBER	DESCRIPTIO
EM167011	Complete PICMASTER-16B System for PIC16C71
EM167012	Complete PICMASTER-16B System for PIC16C71 without Programmer
EM167013	Complete PICMASTER-16C System for PIC16C84
EM167014	Complete PICMASTER-16C System for PIC16C84 without Programmer
EM167015	Complete PICMASTER-16D System for PIC16C5X
EM167016	Complete PICMASTER-16D System for PIC16C5X without Programmer
EM167017	Complete PICMASTER-16E System for PIC16C64
EM167018	Complete PICMASTER-16E System for PIC16C64 without Programmer
EM167019	Complete PICMASTER-16F System for PIC16C74/C73
EM167020	Complete PICMASTER-16F System for PIC16C74/C73 without Programmer
EM177001	Complete PICMASTER-17 System for PIC17C42
EM177004	Complete PICMASTER-17 System for PIC17C42 without Programmer

© 1994 Microchip Technology Inc.

MICROCHIP

PRO MATE™

CMOS Microcontroller Programmer Unit

SYSTEM FEATURES

EPROM Programmer System:

- PRO MATE Programmer unit for the PIC16CXX, PIC17CXX Microcontroller family.
- Operates as a Stand-alone Unit or in Conjunction with a PC Compatible host system.
- READS, PROGRAMS, and VERIFIES in Stand-alone mode.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit
- Communication Via RS-232

SYSTEM DESCRIPTION

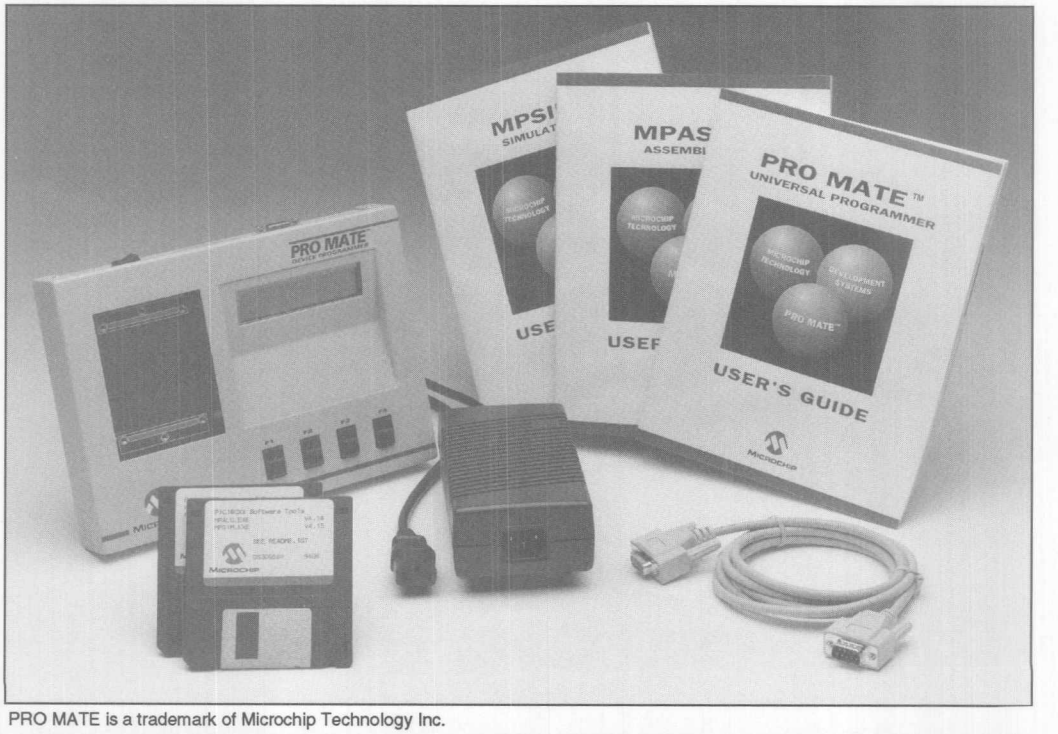
PRO MATE Programmer:

The PRO MATE Programmer system provides the product developer with the ability to program user software into PIC16CXX, PIC17CXX CMOS microcontrollers.

PRO MATE is also supplied with a discrete event software simulator (MPSIM) for PIC16C5X / PIC16CXX and a Universal PIC16/17 Macro assembler.

The programmer unit comes complete with accessories to be used with the PC host computer. Supplied are interface cables and connectors to a standard PC serial port, a universal input power supply unit, and host operating software.

The PRO MATE Programmer will work in either stand-alone mode, or in PC host connected mode. Connected to a PC host, many more features are available to the user.



PRO MATE is a trademark of Microchip Technology Inc.

9

© 1994 Microchip Technology Inc.

STAND-ALONE MODE

Stand-alone mode is useful in situations where a PC may not be available or even required, such as in the field or in a lab production environment. In stand-alone mode the following programming functions are available:

VERIFY

VERIFY performs two functions. For a programmed part, the device in the programming socket will be compared to the program data stored in internal memory. If the data and fuse settings are correct, VERIFIED will be displayed. VERIFY will also confirm that erased parts are blank. A device in the socket will display ERASED if all programmable locations are blank.

PROGRAM

In stand-alone mode, devices inserted into the programmer socket will be programmed with data currently stored in memory. Pressing the PROGRAM key will cause the unit to program and verify both the program memory and the device fuses. If all program successfully, PGM OKAY will be displayed.

READ

A pre-programmed device placed in the programmer socket can be read into the programmer unit by pressing the READ key. Program and fuse data will be read and stored into internal memory. Various options exist with the READ function.

PC HOST CONNECT MODE

The PRO MATE provides a very user friendly user interface which allows complete control over the programming session.

The PRO MATE host software is a DOS windowed environment with full mouse support to allow the user to point and click when entering commands.

The Host Software communicates with the PRO MATE via the serial port of the PC. Any of the four (COM 1-4) ports may be used. The communication is done at 19200 baud to insure fast throughput. Communication will be established with the PRO MATE Device Programmer prior to any transfers taking place.

Serialization is done by generating a serialization file, and then using that file to serialize locations in the PIC16/17 microcontroller. Once a serialization file is generated, it may be used over different programming sessions. Serial numbers are automatically marked as used when a PIC16/17 is programmed successfully with that serial number.

Complete control over the programming environment is also provided. Control over the programming and verify voltage of Vdd insures that the Microcontroller will perform in the desired environment. Programming (Vpp) voltage is also adjustable to insure complete compatibility with future programming algorithms.

© 1994 Microchip Technology Inc.

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16/17 microcontroller product family.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16/17 Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16/17 microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

SALES AND SUPPORT

To order or to obtain information, e.g., on pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

SOCKET PART NUMBER	DESCRIPTION
AC164001	PIC16C54 thru C58A 18- & 28-Lead PDIP Socket Module
AC164002	PIC16C54 thru C58A 18- & 28-Lead SOIC Socket Module
AC164003	PIC16C54/56/58A 20-Lead SSOP Socket Adapter†
AC164010	PIC16C71/84 18-Lead PDIP/SOIC Socket Module
AC164011	PIC16C55/57 28-Lead SSOP Socket Adapter†
AC164012	PIC16C64/C74 40-Lead PDIP Socket Module
AC164013	PIC16C64/C74 44-Lead PLCC Socket Module
AC164014	PIC16C64/C74 44-Lead PQFP Socket Adapter†
AC174001	PIC17C42 40-Lead PDIP Socket Module
AC174002	PIC17C42 44-Lead PLCC Socket Module
AC174003	PIC17C42 44-Lead QFP Socket Adapter

PROGRAMMER PART NUMBER	DESCRIPTION
DV007001	Programmer Kit as described above

Socket modules are sold separately.

† This item is a socket adapter. When using this adapter, a PDIP socket module is also required.

9

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

PICSTART™-16B1

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:

- EPROM Development Programmer unit for the PIC16C5X and selected PIC16CXX Microcontroller family members. Supports PIC16C54, PIC16C55, PIC16C56, PIC16C58A, PIC16C57, PIC16C71, PIC16C84.
- Operates with a PC-compatible host system.
- READS, PROGRAMS, and VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS232 interface cable

Macro Assembler:

- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16CXX microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

"Quick Start" Sample Kit:

- Provides the User / Developer with a sample kit of PIC16CXX parts for initial prototype use.



PICSTART is a trademark of Microchip Technology Inc.
MS-DOS is a registered trademark of Microsoft Corp.

9

© 1994 Microchip Technology Inc.

SYSTEM DESCRIPTION

The PICSTART-16B1 Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16B1 system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C71 or PIC16C84 microcontrollers.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16B1 Development Programmer:

The Microchip device programmer system included in the PICSTART-16B1 Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPASM Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16CXX assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. Since the PIC16CXX architecture is essentially a single tasking microcontroller without interrupts, many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter

Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurances of instructions, register use, and ASCII data.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part number, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DV163003	PICSTART-16B1 DEVELOPMENT SYSTEM

© 1994 Microchip Technology Inc.

PICSTART™-16C

PIC16CXX Low-Cost Microcontroller Development System

SYSTEM FEATURES

EPROM Programmer System:

- EPROM Programmer unit for the PIC16CXX Microcontroller family. Supports the PIC16C64 and the PIC16C74.
- Operates with a PC-compatible host system.
- READS, PROGRAMS, and VERIFIES EPROM Memory.
- PC Host Software provides file display and editing, and transfer to and from Programmer unit.
- Universal power supply
- RS232 interface cable

Macro Assembler:

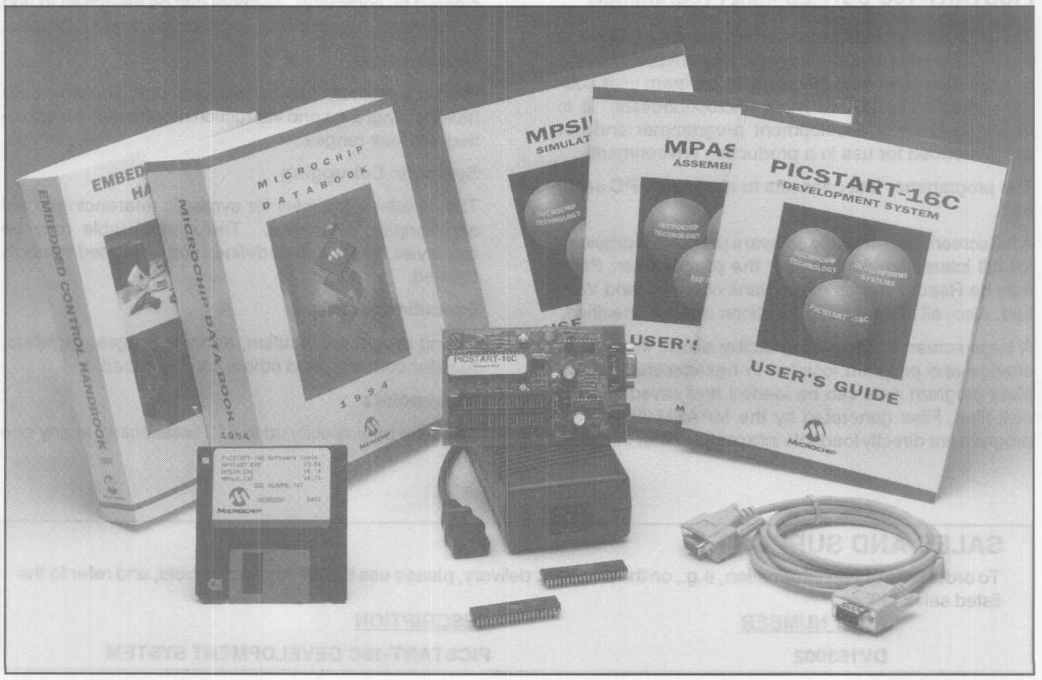
- Provides translation of Assembler source code to object code for all PIC16CXX microcontroller product family.
- Macro-Assembly capability.
- Provides Object files, Listing files, Symbol files, and special files required for symbolic debug with the PIC16CXX Emulator System.
- Output formats: INHX8S and INHX8M.

Simulator:

- Instruction-level Simulator of the PIC16CXX microcontroller product family.
- For PC-compatible systems running the MS-DOS® operating system.
- Full screen simulation user interface.
- Symbolic debugging capability.
- I/O stimulus input capability.

"Quick Start" Sample Kit:

- Provides the User / Developer with a sample kit of the supported PIC16CXX parts for initial prototype use.



PICSTART is a trademark of Microchip Technology Inc.
MS-DOS is a registered trademark of Microsoft Corp.

9

© 1994 Microchip Technology Inc.

PIC16CXX PICSTART System

SYSTEM DESCRIPTION

The PICSTART-16C Development System provides the product development engineer with an alternative low-cost introductory microcontroller design tool set for the PIC16CXX family where full real-time emulation is not required. The equipment in the PICSTART-16C system operates on any PC compatible machine running the MS-DOS/PC-DOS operating system.

Provided in the System is an MS-DOS-based Software Simulator program (MPSIM), a microcontroller EPROM programmer, and a macro assembler program (MPASM).

Sample software programs to be run on the simulator are provided to help the user to quickly become familiar with the development system and the PIC16CXX microcontroller line.

The user need only provide his or her own preferred text editor and the system is ready for development of end products using the PIC16C64 or the PIC16C74.

A "Quick Start" PIC16CXX Product Sample Pak containing user programmable parts is also included.

Microchip provides additional customer support to developers through an electronic Bulletin Board System (BBS). Customers have access to the latest updates in software as well as application source code examples. Consult your local sales representative for information on accessing the BBS.

PICSTART-16C Development Programmer:

The Microchip device programmer system included in the PICSTART-16C Development System provides the product developer with the ability to program user software into PIC16CXX EPROM microcontrollers. It is designed to be a development programmer and not recommended for use in a production environment.

The programmer unit connects to a standard PC serial port.

A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be Read, Programmed, Blank checked, and Verified. Also, all fuses and ID locations may be specified.

A large screen buffer editing facility allows the user to change and program location in hexadecimal. Complete program data can be loaded and saved to DOS disk files. Files generated by the MPASM Assembler program are directly loadable into programmer memory.

MPSIM Simulator:

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging PIC16CXX assembler code.

The simulator is meant for use with smaller projects not requiring precise more extensive development equipment. Since the PIC16CXX architecture is essentially a single tasking microcontroller without interrupts, many applications can be developed by using a simulator program alone.

The MPSIM Simulator has the following features to assist in the debugging of software/firmware for the user.

Program Load/Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display & Alter

Provisions are made to display and alter Program Memory, Register Files, and status register bits. Also simulator information such as cycle times, elapsed time, and step count can be displayed.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurances of instructions, register use, and ASCII data.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, address ranges, registers, register contents, and others can be traced.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DV163002	PICSTART-16C DEVELOPMENT SYSTEM

MS-DOS is a registered trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

MICROCHIP

PICDEM-1

Low-Cost PIC16/17 Demonstration Board

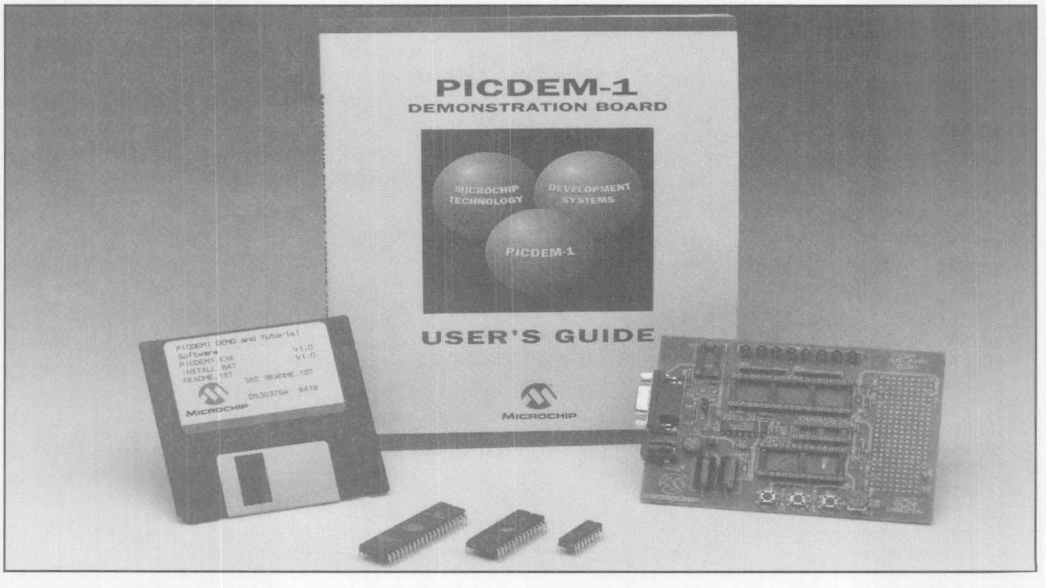
PRODUCT INFORMATION

The PICDEM-1 is a simple board which demonstrates the capabilities of several Microchip microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58), PIC16C71, PIC16C84 and PIC17C42. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. The users can program the samples (one each of PIC17C42, PIC16C71 and PIC16C55) provided with the PICDEM-1, on a PRO MATE™ or PICSTART™ programmer and easily debug/test the sample code, or the user can connect the PICDEM-1 with the PICMASTER emulator and download the sample code to the emulator and debug/test the code. Additionally, a generous 200-hole prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s).

FEATURES:

Hardware:

- 40-pin, 28-pin and 18-pin Precision sockets for all supported microcontrollers.
- On board +5V regulator and filter rectifier for direct input from 9V AC/DC wall adapter.
- RS232 socket and associated hardware for direct connection to RS232 interface.
- 5K pot to simulate analog input for PIC16C71.
- Three push button Key for external stimulus and RESET.
- Eight bright LEDs connect to PortB, help in displaying 8-bit binary values on PortB.
- Socket for "canned" crystal Oscillator.
- Unpopulated holes provided for Xtal connection
- Jumper to disconnect on board RC Oscillator.
- 200-hole prototype area for user's hardware.



PRO MATE and PICSTART are trademarks of Microchip Technology Inc.

© 1994 Microchip Technology Inc.

Software:

- Program for PIC16C71 to demonstrate on-chip A/D features.
- Program for PIC16C84 to demonstrate on-chip EEPROM.
- Program for PIC17C42 to demonstrate on-chip USART.
- Program for PIC16C5X to demonstrate key input capability.
- All demo programs supplied on 3.5" disk,
- Additional programs available on Microchip's BBS.

DOCUMENTATION

- A comprehensive User's Guide with easy to follow step-by-step Getting Started and a Tutorial.
- Schematics for the entire circuit.

SAMPLES

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC17C42
- PIC16C71
- PIC16C55

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163001	LOW-COST DEMONSTRATION BOARD FOR PIC16C5X, PIC16C71, PIC16C84 AND PIC17C42

© 1994 Microchip Technology Inc.

MICROCHIP

PICDEM-2

Low-Cost PIC16CXX Demonstration Board

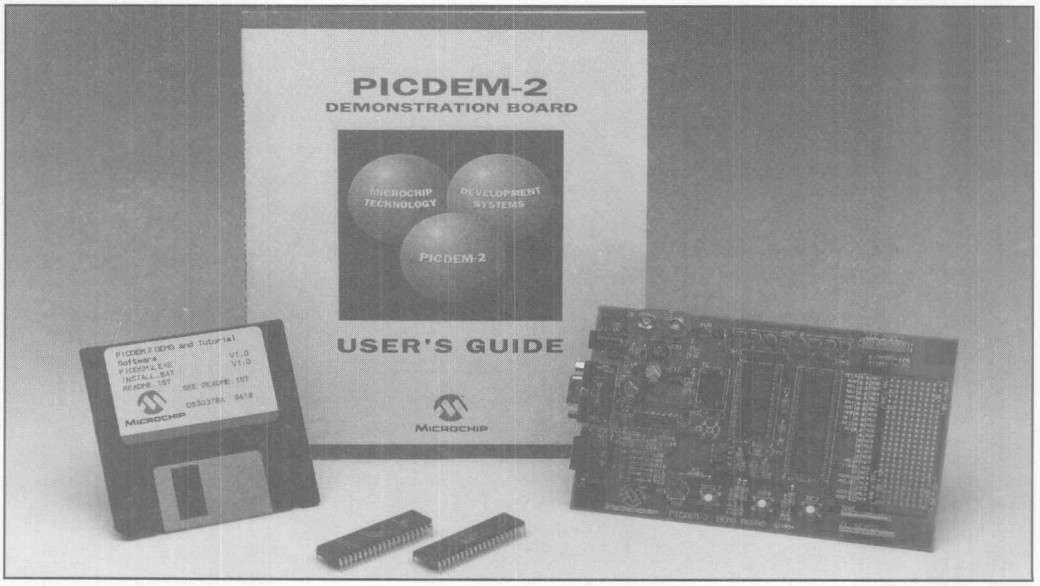
PRODUCT INFORMATION

The PICDEM-2 is a simple board which demonstrates the capabilities of several Microchip microcontrollers, including PIC16C63/64 and PIC16C73/74. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE™ or PICSTART™ programmer and easily debug and test the sample code. The PICDEM-2 is also useable with the PICMASTER™ emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, A generous prototype area is available for user hardware.

FEATURES:

Hardware:

- 40- and 28-pin DIP sockets
- On board +5V regulator for direct input from 9V AC/DC wall adapter.
- RS232C socket and associated hardware for direct connection to RS232C interface.
- 5K pot for analog inputs for the PIC16C73/74
- Three push button keys for external stimulus and RESET.
- Eight bright LEDs connected to Port_B for displaying 8-bit binary values.
- Socket for "canned" crystal oscillator.
- Unpopulated holes provided for crystal connection
- 128 x 8 serial EEPROM.
- LCD module header.
- Keyboard header.
- Unpopulated holes for ACCESS.bus connector.
- Jacks for connection of 9V battery.
- Jumper to disconnect on-board RC oscillator.
- Prototype area for user hardware.



PRO MATE, PICSTART and PICMASTER are trademarks of Microchip Technology Inc.

© 1994 Microchip Technology Inc.

Software:

- Program for PIC16C74 to demonstrate on-chip A/D feature.
- Program for PIC16C64 to demonstrate I²C serial EEPROM usage.
- All demo programs supplied on 3.5" disk.
- Additional programs available on Microchip's BBS.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

SAMPLES:

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC16C64
- PIC16C74

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163002	LOW-COST DEMONSTRATION BOARD FOR PIC16C64, PIC16C73 AND PIC17C74

© 1994 Microchip Technology Inc.

MICROCHIP MPASM UNIVERSAL ASSEMBLER

Universal PIC16/17 Microcontroller Assembler Software

This product brief describes the technical aspects of the PIC16/17 Assembler developed by Byte Craft Limited and distributed by Microchip Technology. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X, PIC16CXX and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the Microchip Universal Emulator System (PICMASTER).

MPASM REQUIREMENTS

MPASM will run on any IBM PC/XT®, AT® or compatible computer running DOS 4.1 or later. The distribution media is 3 1/2", low density (720K) floppies. It is distributed at the root level, and may be executed directly from the floppy.

No special display or ancillary devices are required.

MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.
- Output formats: INHX8S, INHX8M, INHX32 and relocatable objects.

MPASM DIRECTIVE LANGUAGE

MPASM provides a full featured directive language represented by four basic classes of directives:

- **Data Directives** are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- **Listing Directives** control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- **Control Directives** permit sections of conditionally assembled code.
- **Macro Directives** control the execution and data allocation within macro body definitions.

MPASM INSTRUCTION SET

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Data Transfer Operations
- Logical Operations
- Rotate Operations

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.

9

IBM PC/XT and AT are registered trademarks of IBM Corp.

© 1993 Microchip Technology Inc.

MPLINK

MPLINK supports the linking of multiple relocatable objects created by MPASM into a single absolute hex file, suitable for simulating, emulating, and programming. MPLINK supports the hex outputs supported by MPASM and absolute listing file.

MPLIB

MPLIB provides the ability to group several relocatable objects into a logical collection, in one file. Libraries created by MPLIB can be referenced by MPASM. Only those objects required by the linking phase are included in the resulting hex output.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
SW165002	MPSIM/MPASM SIMULATOR AND ASSEMBLER SOFTWARE AND DOCUMENTATION

© 1993 Microchip Technology Inc.

MICROCHIP MPSIM SIMULATOR

PIC16C5X and PIC16CXX Microcontroller Simulator

MPSIM is a discrete event simulator software application designed to imitate operation of the PIC16C5X and PIC16CXX microcontrollers. It allows the user to debug software that will use any of these microcontrollers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPSIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of you project.

MPSIM REQUIREMENTS

MPSIM requires an IBM PC/XT®, AT® or compatible computer running DOS version 3.31 or later. The PC needs a 3 1/2" floppy disk drive and at least 256K of main memory; MPSIM.EXE occupies roughly 150K. Recommended is a hard disk drive with 5 Mb of available storage.

MPSIM SIMULATOR

The MPSIM Simulator program provides the developer with an instruction and limited I/O simulator software program for debugging Microchip microcontroller assembler code.

The simulator is meant for use with smaller projects not requiring precise, more extensive development equipment. Since the PIC16C5X architecture is essentially a single tasking microcontroller without interrupts, many applications can be developed by using a simulator program alone.

The PIC16CXX family supports various peripherals and interrupt strategies. These interrupts can be simulated but certain peripheral functions (such as A/D conversions) are not.

The MPSIM Simulator has the following features to assist in the debugging of software / firmware for the user:

Program Load / Save

Commands exist to load assembled object file programs into simulation memory. Conversely, programs may be saved from program simulation memory back to the PC disk.

Display and Alter

Provisions are made to display and alter Program Memory, Register Files and status register bits. Also, simulator information such as cycle times, elapsed time, and step count can be displayed.

Disassembler

Program memory can be disassembled showing both hexadecimal data and instruction mnemonics for specified address ranges.

Utility Functions

Various utility functions exist which assist the user in operating the simulator. Memory and registers can be cleared by command. Memory can be searched to find occurrences of instructions, register use and ASCII data.

Symbolic Debugging

The simulator provides for symbolic referencing to aid and simplify debugging. The symbol table may be displayed. New symbols defined and unwanted symbols deleted.

Execution and Trace

During program execution, a number of items can be traced. Address ranges, registers and register contents and others.

Breakpoints

The user may specify up to 512 breakpoints at any one time.

Assembler Support

MPSIM supports both the Microchip MPALC and the MPASM Universal Assembler.

IBM PC/XT and AT are registered trademarks of IBM Corp.

© 1993 Microchip Technology Inc.

NOTES:

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
SW165002	MPSIM/MPASM SIMULATOR AND ASSEMBLER SOFTWARE AND DOCUMENTATION

© 1993 Microchip Technology Inc.

MICROCHIP

MP-C

MP-C Code Development System

MP-C CODE DEVELOPMENT SYSTEM FOR PIC16/17

This product brief describes the technical aspects of the **MP-C** Code Development System for PIC16/17 microcontrollers developed by Byte Craft Limited.

The **MP-C** Code Development System is a complete 'C' compiler and integrated development environment for Microchip's PIC16/17 family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the PICMASTER Universal Emulator memory display (emulator software versions 1.13 and later).

MP-C is fast and efficient. You can quickly produce stand-alone single-chip microcontroller applications. These, taken with its other advantages make the Byte Craft **MP-C** Code Development System the first choice in intelligent compiler technology.

MP-C Requirements

The compiler will run on any IBM PC®, PC/XT®, PC/AT® or compatible computer, running DOS 4.1 or later. The distribution media is $3\frac{1}{2}$", low density (720k) floppies. It may be executed directly from the floppy, or copied to your hard drive to execute more conveniently.

MP-C Code Development System Features

MP-C supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and 16-bit PIC17CXX cores. It is a rule-based compiler with expert systems tailored for each of these platforms for optimal efficiency.

The compiler generates executable code directly from the compile process. There is no need for an extra step to assemble code generated by the compiler.

MP-C has the following features to assist in developing PIC16/17 software for specific user applications:

- Provides Object, Listing, Symbol and special files required for debugging with other Microchip Development systems.
- Supports interrupt routines
- Checks source against target hardware definitions
- Generates efficient, tight object code
- Includes a linker and built-in macro assembler
- 'C' enhancements specific to the PIC16/17 families' instruction sets
- Output formats: INHX8S, INHX8M, and INHX32.

MP-C Microprocessor Specific Extensions

The **MP-C** Code Development System includes common 'C' enhancements such as ROM arrays, binary constants and case statements together with functions specific to the PIC16/17 architecture.

- **Binary Constants** of the form *0b0101110* which are logical extensions to the conventional *0x1a3b* style of hexadecimal constants. You may also use *0B* as leading characters.
- **Case Statements** are supported well by the PIC16/17 instruction set and the compiler provides a superset of the standard 'C' case statement. For example, *case 4,5:*, *case '0'..'9'*, and complex case statements are allowed.
- **Processor Specific Functions** that are specific to the PIC16/17 family. For example *NOP()* and *SLEEP()* produce the equivalent PIC16/17 instruction.
- **"At" or @ Extension** allows you to fix a variable to a specific address in memory, for example: *int N @ 0x0C.*

9

SALES AND SUPPORT

The **MP-C** Code Development System is supplied and supported directly by Byte Craft Limited of Waterloo, Ontario.

If you have any questions please contact ByteCraft's technical support personnel at **(519)888-6911**, or fax your questions to **(519)746-6751**.

IBM PC, PC/XT and PC/AT are registered trademarks of IBM Corporation.

© 1994 Microchip Technology Inc.

NOTES:

© 1994 Microchip Technology Inc.

MICROCHIP

*fuzzy*TECH®-MP

*fuzzy*TECH-MP for PIC16/17

*fuzzy*TECH-MP FOR MICROCHIP PIC16/17

This product brief describes the technical aspects of the *fuzzy*TECH-MP Fuzzy Logic Development System for PIC16/17 microcontrollers developed by INFORM™ Software Corporation specifically for Microchip Technology.

The *fuzzy*TECH-MP Development System comes in two versions. The first, the Explorer, contains everything you need for a comprehensive working knowledge about fuzzy-logic system design. It is easy-to-use, all graphic editors and tools guide you step-by-step through the development phases of fuzzy systems. The Explorer supports two input variables and one output variable.

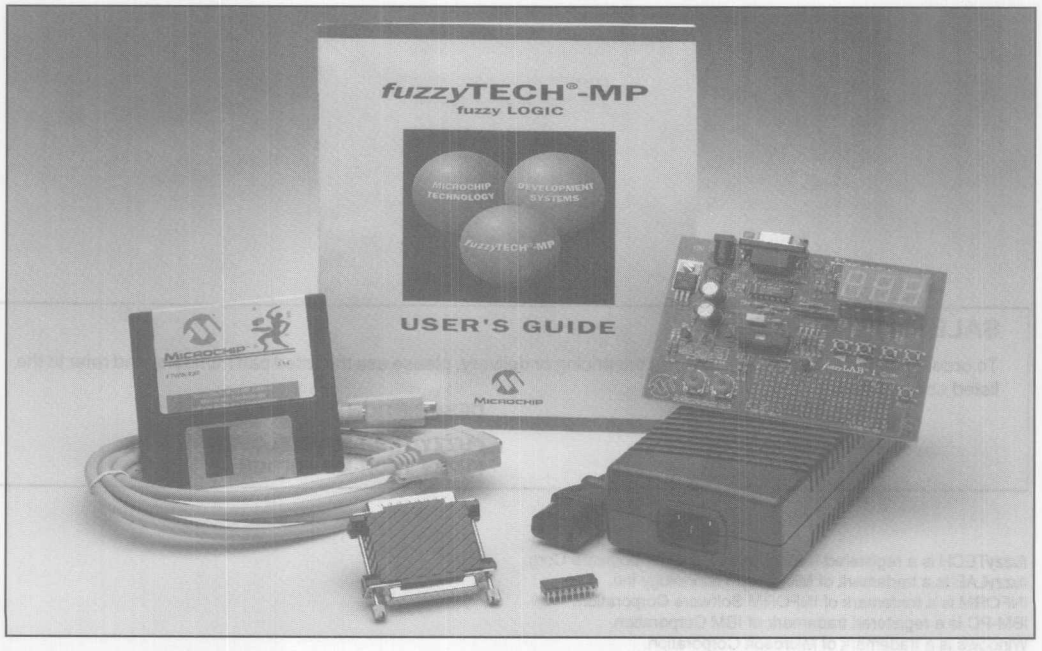
The full-featured *fuzzy*TECH-MP edition offers all of the capabilities of the Explorer, *plus* it has the additional flexibility of eight input variables and four output variables for designing more complex systems. The full features are enabled with a parallel key lock.

Included in both versions is *fuzzy*LAB™, a fully functional demonstration board, to give customers hand-on experience with fuzzy logic systems implementation. fuzzyLAB is a simple heating thermostat consisting of a PWM-controlled resistor configured to heat a thermistor to a preset temperature. Using the two fuzzy algorithms provided, a designer can set a target temperature and observe the thermostat response to the set point.

Both systems generate assembly code compatible with the MPASM, Microchip's Universal Assembler, that can be integrated into your application. Examining this code provides you with further insights into the fabrics of fuzzy logic systems.

*fuzzy*TECH-MP System Requirements

*fuzzy*TECH-MP will run on any IBM PC® (386 or higher) or compatible computer, running DOS 4.1 or later, and Microsoft® Windows™ 3.0 or later. Because *fuzzy*TECH-MP makes extensive use of graphics, a color graphic monitor (VGA) is required, and higher resolutions of 800 x 600 or 1024 x 768 are recommended.



9

© 1994 Microchip Technology Inc.

What is Fuzzy Logic?

Fuzzy logic is a technology that enhances mode-based system designs using both intuition and engineering heuristics. Fuzzy logic uses elements of everyday language to represent desired system behavior, thus circumventing the need for rigorous mathematical modeling.

It is an efficient way of designing, optimizing and maintaining highly complex systems transparently.

Fuzzy Logic Applications

Fuzzy logic finds its home in unique applications:

- When no adequate mathematical model for a given problem is readily apparent.
- When non-linearities, time constraints or multiple parameters exist.
- When engineering know-how about the given problem is available or can be acquired during the design process.

The *fuzzy*TECH-MP Implementation

*fuzzy*TECH-MP provides the following standard features:

- Windows Compatible with full graphical user interface
- 8-Input variables (2 for the Explorer version)
- 4-Output variables (1 for the Explorer version)
- 8-Bit resolution on input and output variables
- 16-Bit computation resolution for the PIC16CXX and PIC17CXX microcontrollers
- No theoretical limit on rules, antecedents and linguistic conjunctions (chip limitations will place a practical limit on these)
- MAX-MIN and MAX-DOT inference methods
- CoM and MoM defuzzification methods
- MPASM Compatible
- PICMASTER Compatible

*fuzzy*TECH-MP

*fuzzy*TECH-MP is available directly from Microchip Technology and its authorized distributors. Contact your local sales office for more information.

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

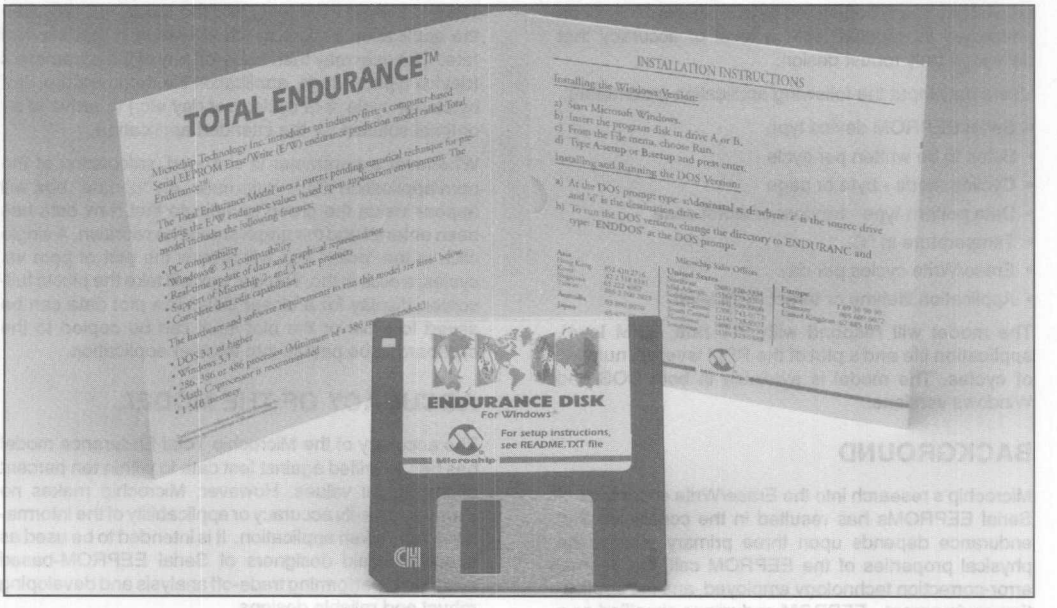
PART NUMBER	DESCRIPTION
DV005001	FUZZY TECH-MP EXPLORER
DV005002	FUZZY TECH-MP EDITION

*fuzzy*TECH is a registered trademark of INFORM Software Corp.
*fuzzy*LAB is a trademark of Microchip Technology Inc.
INFORM is a trademark of INFORM Software Corporation.
IBM-PC is a registered trademark of IBM Corporation.
Windows is a trademark of Microsoft Corporation.
Microsoft is a registered trademark of Microsoft Corporation.

© 1994 Microchip Technology Inc.

MICROCHIP TOTAL ENDURANCE™

Microchip Serial EEPROM Endurance Model



FEATURES

- IBM® PC compatibility
- Windows™ 3.1 or DOS 3.1 compatibility
- Automatic or manual recalculation
- Real-time update of data
- Full-screen or windowed graphical view
- Hypertext on-screen help
- Key or slide-bar entry of parameters
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk Endurance Tutorial

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows 3.1
- 1MB memory
- '386 or '486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip 3-wire 93CXX/93LCXX Series
- Microchip 4-wire 59C11

9

IBM PC is a registered trademark of IBM Corp.
Windows is a trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

DESCRIPTION

Microchip's revolutionary Total Endurance disk provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. This advanced software model (with a very friendly user interface) eliminates time and guesswork from Serial EEPROM-based designs by accurately predicting the device's performance and reliability within a user-defined application environment. Design trade-off analysis which formerly consumed days or weeks can now be performed in minutes...with a level of accuracy that delivers a truly robust design.

Users may input the following application parameters:

- Serial EEPROM device type
- Bytes to be written per cycle
- Cycling mode - byte or page
- Data pattern type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application lifetime or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level vs. number of cycles. The model is available in both DOS and Windows versions.

BACKGROUND

Microchip's research into the Erase/Write endurance of Serial EEPROMs has resulted in the conclusion that endurance depends upon three primary effects: the physical properties of the EEPROM cell, the internal error-correction technology employed, and the application environment. EEPROM endurance specified as a "typical" value in device data sheets must therefore be evaluated on a case-by-case basis, taking into account the manner in which the device will be used in the application. The Microchip Total Endurance software applies the user-defined application parameters to a complex mathematical model in order to emulate the EEPROM's performance and reliability in the system.

USING THE MODEL

The user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

ACCURACY OF THE MODEL

The accuracy of the Microchip Total Endurance model has been verified against test data to within ten percent of the actual values. However, Microchip makes no warranty as to its accuracy or applicability of the information to any given application. It is intended to be used as a guide to aid designers of Serial EEPROM-based systems in performing trade-off analysis and developing robust and reliable designs.

© 1994 Microchip Technology Inc.

MICROCHIP

DESIGNER'S KIT

Microchip Serial EEPROM Designer's Kit



FEATURES

- Includes everything necessary to begin developing Serial EEPROM-based applications
- Microchip *Total Endurance*™ software model
- Microchip *SEEVAL*™ evaluation and programming board
- Microchip *SEEVAL*™ software
- Microchip Serial EEPROM handbook
- Microchip Serial EEPROM sample pack
- RS232 serial cable
- Power supply

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows™ 3.1
- VGA monitor
- '386 or '486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip *Smart Serial*™ 24XX65
- Microchip 3-wire 93CXX/93LCXX series
- Microchip 4-wire 59C11

9

Total Endurance, SEEVAL, and Smart Serial are trademarks of Microchip Technology Inc.
IBM PC is a registered trademark of IBM Corp.
Windows is a trademark of Microsoft Corp.

© 1994 Microchip Technology Inc.

Serial EEPROM Designer's Kit

DESCRIPTION

Now designers of Serial EEPROM-based applications can enjoy the increased productivity, reduced time to market, and the ability to create a rock-solid design that only a well-thought-out development system can provide. Microchip's new Serial EEPROM Designer's Kit includes everything necessary to quickly develop a robust and reliable Serial EEPROM-based design and greatly reduce the time required for system integration and hardware/software debug.

The **TOTAL ENDURANCE SOFTWARE MODEL** enables designers to quickly choose the best Serial EEPROM for the specific application and perform trade-off analysis with voltage, temperature, write cycle and other system parameters in order to achieve the desired Erase/Write endurance (specific ppm rate) or product lifetime. Total Endurance is the new standard of excellence in understanding and predicting the Erase/Write endurance of Serial EEPROMs. An online endurance tutorial is included, along with hypertext help files.

Microchip's **SEEVAL SERIAL EEPROM EVALUATION AND PROGRAMMING SYSTEM** will accept any Microchip Serial EEPROM in DIP package and enable the designer or system integrator to read, write, or erase any byte or the entire array. SEEVAL also provides the following advanced features to aid in system integration and debug:

- Program special user functions like *Smart Serial* configurations
- Hexadecimal display of array contents
- Pre-set or user-defined repeating patterns
- User-configurable functions like continuous read/write, programmable delay, etc.
- Upload/download files between the Serial EEPROM and disk

Another industry first, the **MICROCHIP SERIAL EEPROM HANDBOOK** provides a plethora of information crucial to the designers of Serial EEPROM-based systems. Along with data sheets on Microchip Serial EEPROMs, this resource provides application notes regarding Erase/Write endurance, interfacing with different protocols and many, many others. A cross-reference and selector guide are also included, plus article reprints and qualification reports on Microchip Serial EEPROMs.

USING SEEVAL AND TOTAL ENDURANCE

Both software packages can be loaded from Windows by choosing FILE RUN and entering SETUP.EXE from the Program Manager. The applications will install themselves; then a double mouse-click will start either application. The first step in either program is to select a device from the device list.

In Total Endurance, the user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

In **SEEVAL**, the user may choose to load a file from disk to program the Serial EEPROM, or read data from the EEPROM and save it to disk. The screen displays the contents of a software buffer. The buffer may be manipulated before programming data to the Serial EEPROM, or data can be written to the Serial EEPROM directly online. An area of memory can be highlighted (selected) and programmed with a predefined pattern or user-specified pattern. Alternately, the entire device can be programmed with any repeating pattern.

Both SEEVAL and Total Endurance allow the user to save any configuration as default. This configuration (device and application settings) will then automatically load at boot time.

© 1994 Microchip Technology Inc.

SECTION 10
SELECTED READING OF MICROCHIP ARTICLES

Selected Reading ..... 10 - 1

© 1994 Microchip Technology Inc.

Selected Reading

"Berechnung der Lebens – erwartung von EEPROMs," *Elektronik Industrie*, Germany, November 1993, Page 60.

"Brainwave Invention – Brainwave Enhancer," *Electronic Times*, UK, September 9, 1993, Page 2.

Burghardt, M. "In-Circuit-Programmierung von OTP-Controllern," *Elektronik Industrie*, March 1994, Page 56.

Bursky, Dave. "Simple Microcontroller-based Scrambling Scheme Blocks Video Cassette," *Electronic Design (ED)*, September 16, 1993, Page 48.

Cates, Ron. "Advanced 0.9-micron process enhances performance of Microchip PIC16C54A 8-bit OTP microcontrollers," Business Wire, U.S.A., March 29, 1994.

"Chips: New Family of "Smart Serial" EEPROMs Marks a Turning Point in Non-Volatile Memory Technology," *EDGE (EDOA)*, Vol. 8, No. 251 É3;É3, May 17, 1993.

"Circuit de controle de charge de batterie," *Electronique Techniques et Industries*, March 1994, Page 31.

Danielli, G. "Microcontrollore Mid-range a otto bit," *Elettronica*, Italy, No. 174, February 28, 1994.

Dunn, Darrell. "Fast MCU Sports On-Chip EEPROM – Applications Range From Security To Smart Credit And Transit Cards" *Electronic Buyers' News*, U.S.A., Issue 847, page 21, March 29, 1993.

Dunn, Darrell. "MCUs Go Submicron," *Electronic Buyers' News*, U.S.A., April 18, 1993.

Dunn, Darrell. "Pumping Up EEPROMsl," *Electronic Buyers' News*, U.S.A., Issue 854, page 10, May 17, 1993.

Dunn, Darrell. "Low Voltage Mouse Control," *Electronic Buyers' News*, U.S.A., Issue 859, page 23, June 21, 1993.

Dunn, Darrell. "Microchip Ups Performance," *Electronic Buyers' News*, U.S.A., Issue 871, page 12, September 20, 1993.

Dunn, Darrell. "Big Memory, Small Package," *Electronic Buyers' News*, U.S.A., Issue 875, page 33, October 18, 1993.

Eady, Fred. "PIC Microcontroller Programmer," *Electronics Now*, January 1994, Page 35.

Eady, Fred. "PIC Microcontroller Programmer," *Electronics Now*, February 1994, Page 48.

"EEPROM a valore aggiunto," *EO News*, September 25, 1993, Page 22.

"E2PROM: De la série haute densité au microcontroleur RISC," *Industromique*, September 1993.

"EEPROM Serie 64 Kbits a Bloc Haute Endurance," *Electronique*, September 1993.

"Eine Million Lösch-und Schreibzyklen," *Elektronik Journal*, August 1993.

"8-Bit Mikrocontroller mit 25 MHz," *Elektronik Informationen*, March 1994.

"Embedded-Control Prowess," *Electronic Buyers' News*, U.S.A., Issue 849, page 25, April 12, 1993.

Frank, Steven and Stegmüller, Manfred. "Intelligenter Positionierantrieb," *Design & Elektronik*, February 22, 1994, Page 62.

"Fuzzy-Logik für Standard MCUs," *Elektronik Revue*, January/February 1994, Page 3.

"Gains Noted In Oscillator Biz," *Electronic Buyers' News*, U.S.A., Issue 845, page 23, March 15, 1993.

"IC überwacht und steuert Batterieladung," Elektronik Informationen, January, 1994, Page 26.

"Ingenious micro," *What's New in Electronics*, March 1994.

Knack, Kella. "Are ASSPs The Wave of The Future?" *ASIC & EDA*, March 1994, Page 10.

"Les Microcontroleurs PIC16CXX de Microchip (3)," *E Radio Plans*, January 1994, Page 31.

"Microchip Adds Fast EEPROMs," *Electronic Buyers' News*, U.S.A., Issue 842, Page 19, February 22, 1993.

10

© 1994 Microchip Technology Inc.

Selected Reading

"Microchip Intros TSOP Option," *Electronic Buyers' News,* March 15, 1993, Page 23.

"Microchip puts microcontrollers in SSOP format," *Electronic Engineering Times*, U.S.A., Issue 735, Page 24, March 1, 1993.

"Microchip Rolls 32K-bit EEPROMs Into Volume," *Semiconductor Industry & Business Survey*, November 8, 1993.

"Microchip Tech rewrites the E(2)PROM," *Electronic Engineering Times*, May 10, 1993, Page 10.

"Microchip Technology Sins New 8-bit MCU," *Electronic News*, April 4, 1994, Page 54.

"Microchip Technology Teknik Egitim Semineri," *Future's Technologies*, January 1994, Page 99.

" Microcontroller upgrade includes reduced power requirements – Arizona Microchip," *Electronic Times (ELTIM)*, UK, May 26, 1994, Page 33.

"Microcontroller der PICSEE-Familie," *Elektronik Industrie*, October 1993.

Monticelli, Lorenzo. "EEPROM: Seriale ma inteligente," *Elettronica*, Italy, No. 163, October 1993.

Nickerson, Jim. "The 8-bit microcontroller is everywhere," *Electronic Engineering Times*, February 1, 1993, Page 53.

Nolan, Jim. "A Solution for Intelligent Battery Management,"*Electronic Product Design*, IMLGroup, Vol. 15, Issue 4, April 1994, Page 49.

Nolan, Jim. "What is Intelligent Battery Management?" Proceedings of the *First annual Portable By Design Conference*, Penton Publishing, February 14-18, 1994, Page BT-12-BT-19.

"PIC: Peripheral Interface Controller," *Computer Design,* December 1993.

"PICSEE in control with low power," *Electronics Times*, October 14, 1993.

Rose, M. "PIC-Programmer, Hard-une Software zur Programmierung von PIC-Mikrocontrollern," *Elektor*, German, No. 189, March 1994, Page 8.

Rose, M. "Programmateur de PIC*," *Elektor*, French, No. 189, March 1994, Page 28.

"Smallest Package for Microcontroller," *Electronic Buyers' News*, January 18, 1993, Page 26.

"Smarte batteripakker," *Elektronik & Data*, October 1993.

"Smart serial memory operates at 1.8V," *Electronic Engineering Times*, Issue 770, page 8, November 1, 1993.

"Tiny EEPROM Package Suits PCMCIA Cards," *Electronic Design*, September 16, 1993, Page 182.

"Two-wire serial E(2)EROMs debut," *Electronic Engineering Times*, September 6, 1993, Page 18.

Tyson, Thomas J. "Serial EEPROMs vs. Parallel Solutions," *Electronic Product Design*, January 1994, Page 43.

Weeder, Terry J. "Take back control of your telephone," *Electronics Now*, Vol. 65, February 1994, Page 33.

Weiss, Ray. "Wanted: low-cost, small 8-bit µCs to take on the dirty work," *EDN*, April 14, 1994, Page 57.

Willett, Hugh G. "MCUs Step Into The Sunlight," *Electronic Buyers' News*, U.S.A., Issue 871, page 282, September 20, 1993.

Wilson, Ron. "Designers probe uncharted 0.9--V realm," *Electronic Engineering Times*, U.S.A., Page 10, May 10, 1993.

Wilson, Ron. "Microchip Tech rewrites the E–2PROM," *Electronic Engineering Times*, U.S.A., Issue 745, Page 48, April 4, 1994.

Wilson, Ron. "Microchip offers speedy 8-bit controller," *Electronic Engineering Times*, U.S.A., Issue 764, September 20, 1993.

© 1994 Microchip Technology Inc.